LightShip Lens™ User's Guide

Version 1.0

PILOT
EXECUTIVE SOFTWARE
40 BROAD STREET • BOSTON, MASSACHUSETTS 02109 • (617) 350-7035

Table of Contents

About This Manual

Chapter 1 Getting Started

Chapter 2 Learning LightShip Lens

Chapter 3 Selecting the Database Source

Chapter 3 Selecting the Database Source *(continued)*

Chapter 4 Displaying The LightShip Lens Data

Chapter 5 Using SQL Select Statements

Chapter 5 Using SQL Select Statements *(continued)*

About This Manual

The *LightShip Lens User's Guide* describes how to create LightShip Lens™ displays in LightShip™ document objects.

Chapter 1: Getting Started describes LightShip Lens and tells you how to start and quit from it. If you are already acquainted with LightShip and want to begin quickly, reading Chapter 1 may be enough to get you started.

Chapter 2: Learning LightShip Lens is a tutorial that shows you how to create a LightShip Lens display. It also shows you how to pass parameters from LightShip to Lens to change the display dynamically when running the application.

Chapter 3: Selecting the Database Source describes how to select and change the source of data for the data cache. This chapter explains the commands in the File and Database menus.

Chapter 4: Displaying the LightShip Lens Data describes how to select the fields and conditions for displaying data. This chapter explains the Conditions, Results, and Sort commands.

Chapter 5: Using SQL Select Statements explains the basics for entering SQL Select statements based on the database systems that LightShip Lens supports.

This manual assumes that you are familiar with Microsoft® Windows™ and with LightShip. LightShip Lens menu commands and dialog boxes follow Windows and LightShip conventions. See the *LightShip User's Guide* to review basic operations.

Conventions Used in This Manual

We use some standard formatting conventions in this manual.

- Menu commands usually follow the menu name they are on. For example:

  1. Choose File Open.

  means "choose the Open command on the File menu."

- A procedure that is about to be described is shown in bold characters next to a shadowed square (❑). Procedures are shown as numbered steps.

- Examples are shown in bold monospaced characters.

- Names that you need to substitute are shown in angle brackets (<>). For example, **<pathname>** means "type in the drive and directory appropriate for your site."

- New terms, action names, variable names, and book titles are italicized.

☞ **Important** Important information, notes, and performance hints are shown next to a hand to make them stand out.

Chapter 1

Getting Started

This chapter describes the basic concepts for understanding and using Pilot's™ LightShip Lens™ most effectively. It explains what LightShip Lens is, how it works, and how to start and quit from the program. It also describes the menus and commands on the menu bar. To install LightShip Lens, refer to the release notes.

What is LightShip Lens?

LightShip Lens is a product for retrieving data in LightShip™ applications. LightShip Lens's simple "pick and choose" interface lets you retrieve and display data without needing to know syntax rules associated with these databases:

- Oracle®
- dBASE®II, dBASE III, and dBASE IV and compatibles with support for NDX and MDX indexes
- ASCII text files, including character-separated files and fixed format files
- Microsoft® Excel™ XLS spreadsheet files
- IBM® DB2®; requires MicroDecision Ware's Database Gateway™ software
- Sybase® and Microsoft® SQL Server™ products

- Sybase for Unix and VMS; requires Windows™ 3.0-compatible network libraries from Sybase
- Netware® SQL™
- Paradox™

The Lens data appears in LightShip document objects. You can perform any standard LightShip operation on a document object with a LightShip Lens source such as formatting and overlaying hotspots.

You can use LightShip variables to pass values from LightShip to Lens. Lens allows LightShip variable substitution in any text box. Figure 1-1 has a non-displayed hotspot over the soft drink products; it changes the Lens display to reflect sales figures for a selected product.

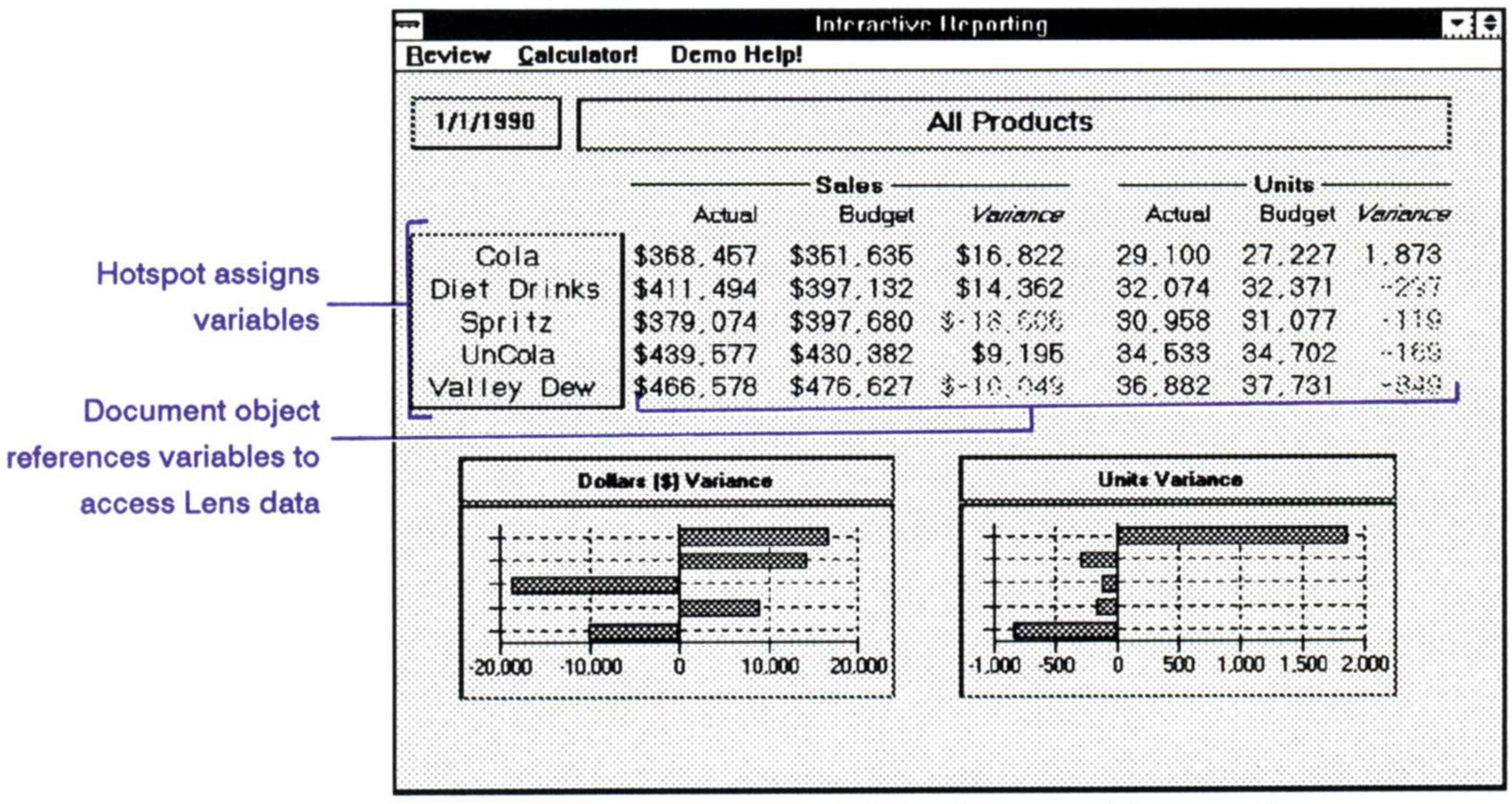

Figure 1-1 LightShip Screen with Lens Data

How Lens Stores Data

Lens lets you select data from a database source and store it in memory. This data is called a *data cache*. Accessing data from the data cache is faster and more manageable than accessing data from the database source; it contains only the data you need to analyze and build an application. Lens works exclusively with the data cache instead of repeatedly accessing the database source.

Loading Data into the Data Cache

Lens loads the data cache the first time that a document object references it during a LightShip session. The data cache remains in memory and can be used by other document objects. Lens can maintain a number of data caches simultaneously. It unloads a data cache at these times:

- When you exit from the LightShip application
- When Lens must unload it to accommodate other data caches in memory
- When you exceed the maximum number of data caches
- When you perform a Calculate Now action in LightShip

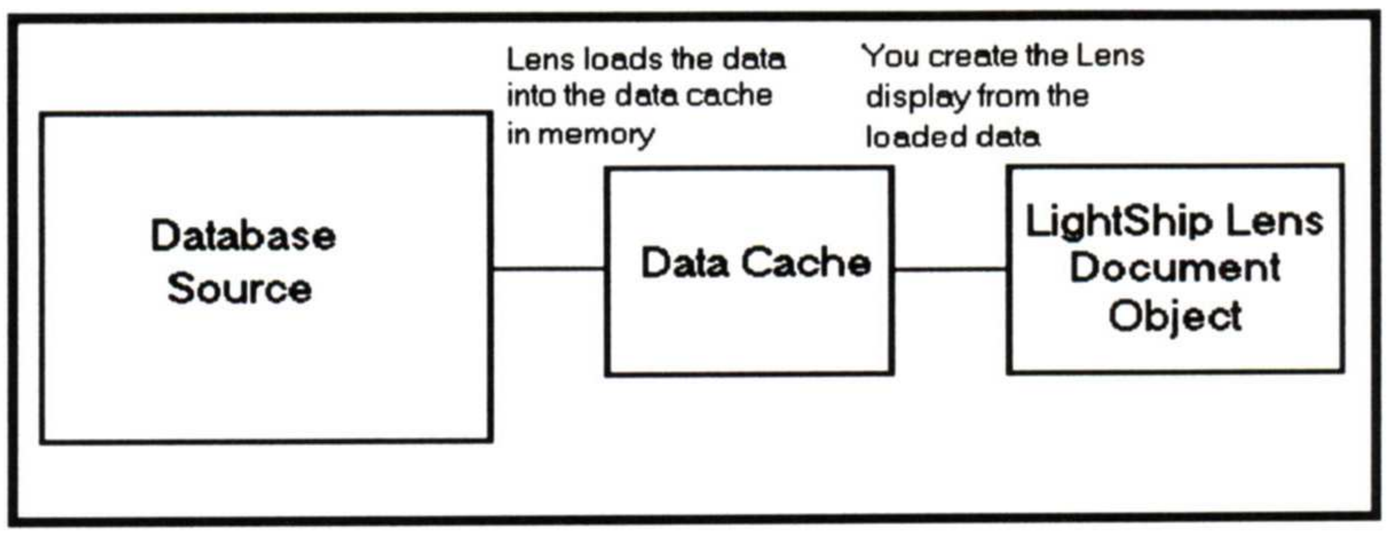

Figure 1-2 The Lens Data Cache

To limit the amount of data in the data cache, you can:

- Load specific fields and conditions using the Database Load commands.
- Type an SQL statement to describe complex SQL selection criteria.

Saving the Data Cache to a File

Optionally, you can save a data cache to a file, called an LSC file. Often, the most efficient design is to create a single LSC file that contains all the data for a particular application. All the document objects in the application can then use the same LSC file as their source.

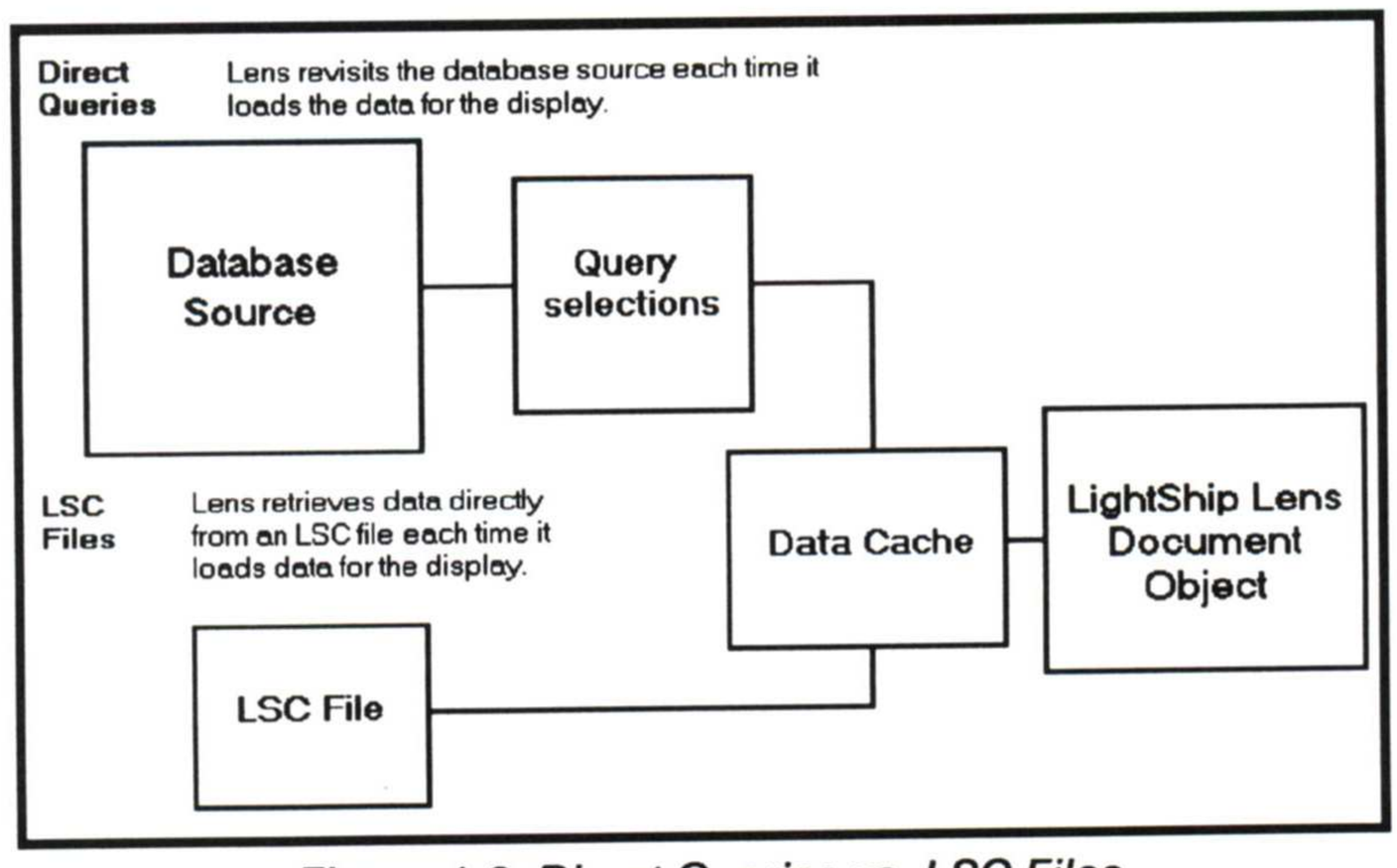

Figure 1-3 Direct Queries vs. LSC Files

If data changes in the database source, you can update an LSC file using:

- The Database Update Data File command.
- A DOS command in Windows.
- A *DOS Command* action in LightShip.

Querying Directly

Here are a few situations in which using a direct query might be a better choice than an LSC file:

- When an application displays data that frequently changes in the database source. For example, if the data changes hourly or even daily, use a direct query to update the data cache directly from the database.

- When a document object displays a few specific records, such as names and addresses from a personal cardfile application.

How Lens Displays Data

To display data from the data cache, you can select specific fields and conditions. You can use LightShip variable references in place of any field names to create a display that changes dynamically. Data is displayed in columns and rows, arranged by one or more dimensions (indexes).

Figure 1-4 shows a Lens window that displays two dimension fields: PRODUCT displays values down the first column and CITY displays values across the top row.

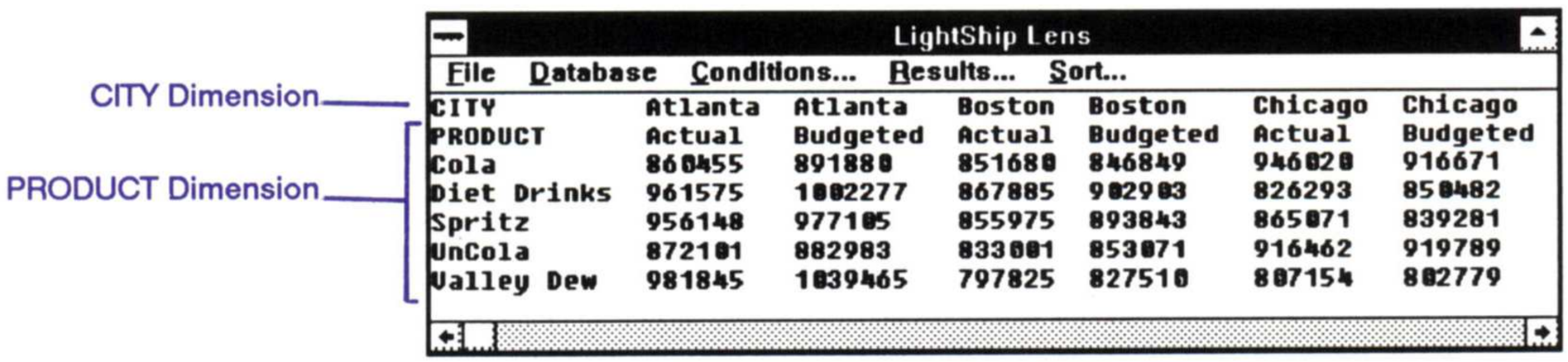

Figure 1-4 Displaying Data in Lens

Resource Considerations

Follow these guidelines when you create a Lens display:

- Each dimension requires that the number of unique values multiplied by the character size of the field be less than 64,000. For example, a dimension field that is 20 characters long allows 3,200 unique values.

- Your system needs enough memory to accommodate the data you want to load. To view the amount of available memory, use the Windows Program Manager Help About Program Manager command. Generally, the amount of memory must be about 20% of the space required to store the file or table on disk.

- Each data cache can contain up to 50,000 records.
- If performance is slow because the software is performing many disk operations, you may need more RAM to dramatically improve performance.

Optimizing Lens's Processing Capabilities

To make the best possible use of Lens's processing capabilities, keep these guidelines in mind:

- Before you load the database source, exclude unnecessary data to save space in memory. You can limit the fields and the categories of data to load.

  For example, assume that you are a regional manager for the Northeast and need sales figures for your area only. You can load data for the Northeast Region only. In addition, if you only need to work with City, Product, and Actual Sales, you can load only these fields.

- While excluding unneeded data, be sure to include all the information with which Lens will need to work.

  For example, assume again that as the Northeast regional manager you only want to display your own sales figures. However, you are also interested in displaying the variance between your sales and sales in other regions. Even though you do not intend to display other regions' sales figures, the query must include the data from the other regions to make the variance calculations possible.

- Use LSC files instead of direct queries whenever possible because they are more efficient.
- Use the same LSC file as the source for all related document objects.
- Avoid using Float Numeric unless the data exceeds six digits or contains numbers to the right of the decimal point. (Turn Float Numeric on and off by choosing the Database Load Fields command.)

Starting Lens

❑ **To start Lens:**

1. Create or select a document object in LightShip.

2. Choose Document Source LightShip Lens.

   The LightShip Lens window appears.

   If the document object is new or if it currently displays data from a non-Lens source, the Lens window is empty and only the File menu is active on the menu bar.

   If the document object already contains Lens data, it appears in the window and the full menu bar is active.

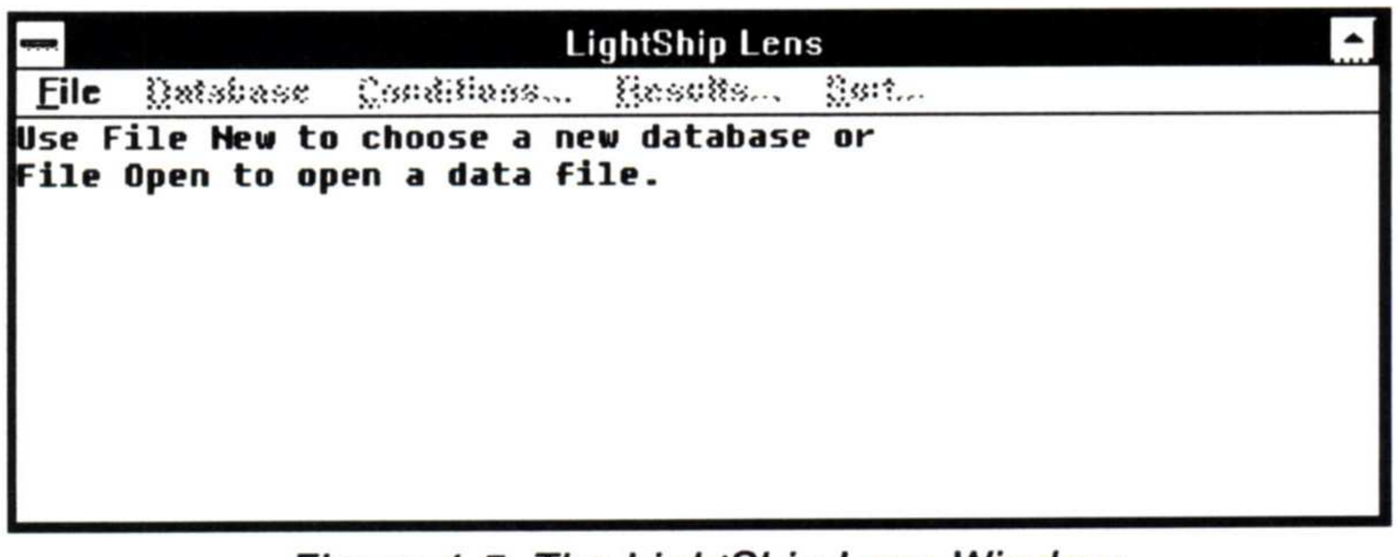

Figure 1-5 The LightShip Lens Window

☞ **Note** If LightShip Lens does not appear on the Document Source menu, the installation was not performed correctly.

The Lens Menu Bar

Figure 1-6 shows the menus and commands on the Lens menu bar.

File	Database	Conditions	Results	Sort
New	Source			
Open	Load Fields			
Save	Load Conditions			
Save As	Update Data File			
Exit/OK				
Exit/Cancel				

Figure 1-6 The Lens Menus

File Menu

New selects a new database source and query and cancels all existing load specifications.

Open opens an existing LSC file as the database source and starts a new query.

Save saves the data cache to the currently open LSC file. If no LSC file is currently open, it performs as though you selected Save As and lets you save the data cache to any specified LSC file.

Save As saves the data cache to the specified LSC file.

Exit/OK exits Lens and saves the current display selections.

Exit/Cancel exits Lens and discards all the display selections made during the current Lens session.

Database Menu

Source changes the current database source and retains all existing load specifications.

Load Fields specifies the fields to load from the database source.

Load Conditions specifies the conditions for limiting the data to load from the database source.

Update Data File updates the currently open LSC file with data from the source.

Conditions, Results, and Sort Commands

Conditions specifies the conditions to limit the current Lens display.

Results selects the fields and arrangement of data for the Lens display.

Sort specifies a sort order other than the default to display dimension values.

Quitting Lens

❑ **To quit Lens and save the changes you made to the current Lens display:**

Choose File Exit/OK.

❑ **To quit Lens and cancel any changes you made to the current Lens display:**

Choose File Exit/Cancel.

Chapter 2

Learning LightShip Lens

This chapter is a tutorial that creates the screen shown in Figure 2-1.

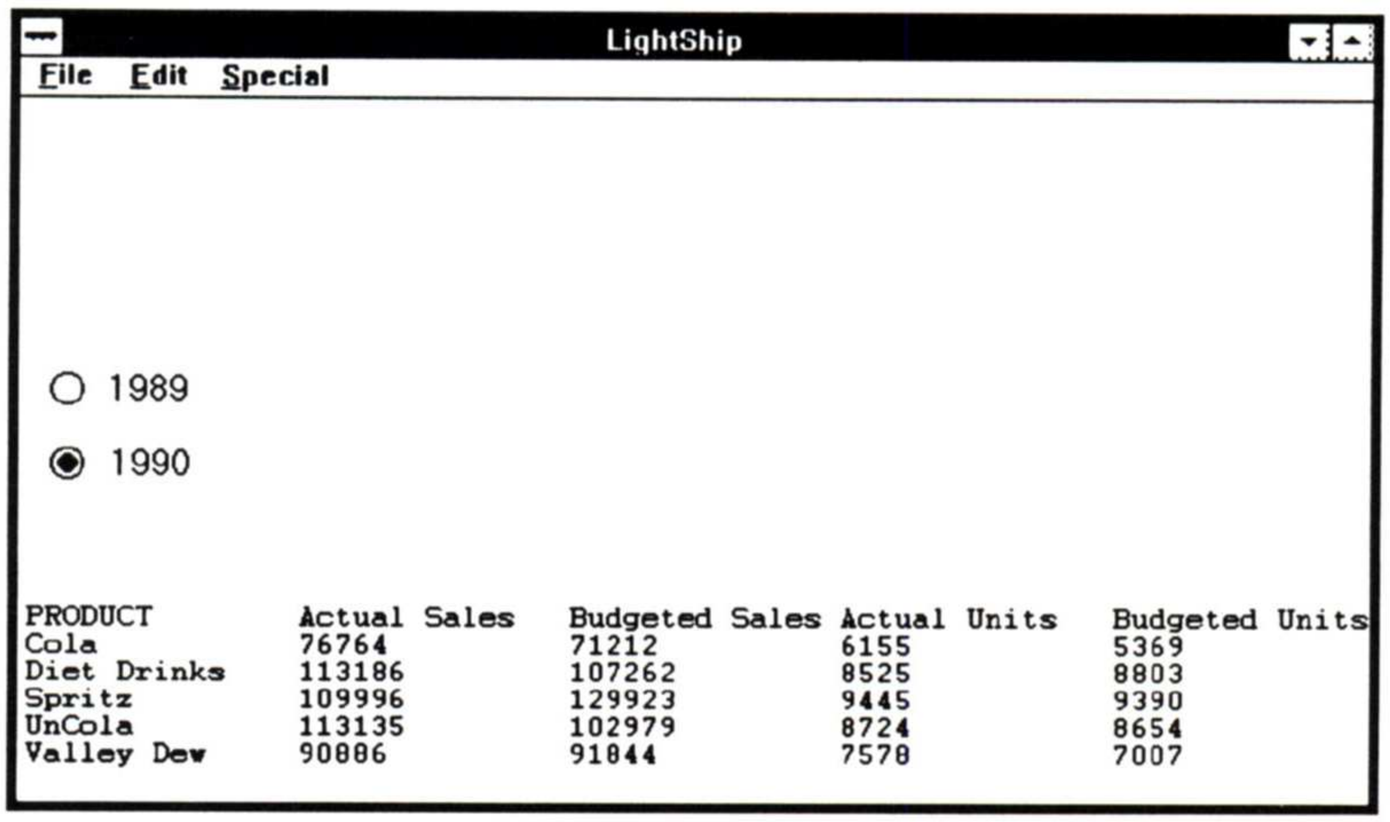

Figure 2-1 The Tutorial Screen

The screen includes a document object that displays soft drink sales by product for the Northeast region of the country. It also includes two hotspots, labeled "1989" and "1990" that change the display to show data for the selected year only.

The tutorial has four parts:

- "Specifying the Database Source" selects a dBASE file and loads only the data for the Northeast region into the data cache. Then it saves the data cache to an LSC file. The DRINKS.DBF file is included with the LightShip demonstration files.

- "Creating the Display" specifies fields and a variable-driven condition for displaying the Lens data.
- "Displaying Lens Data Dynamically in LightShip" creates the **1989** and **1990** hotspots in LightShip.
- "Browsing Through the Application" takes you through the application at the Browse level.

Starting LightShip and Lens

These steps start LightShip and create a document object whose source is LightShip Lens.

❑ **To start:**

1. Start LightShip.
2. Choose Tool Document and drag the mouse to create a document object.

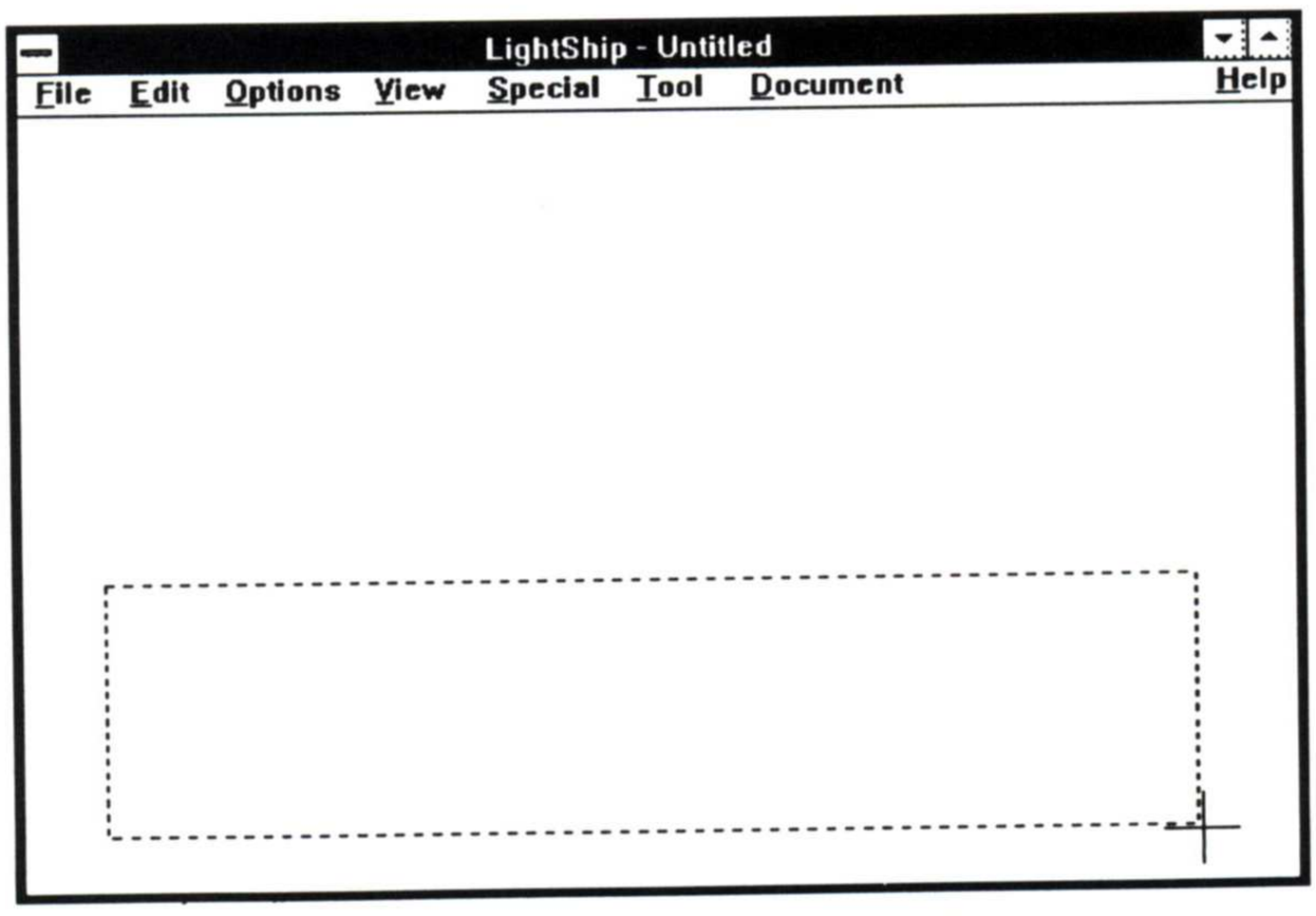

Figure 2-2 Creating a Document Object

3. Choose Document Source LightShip Lens.

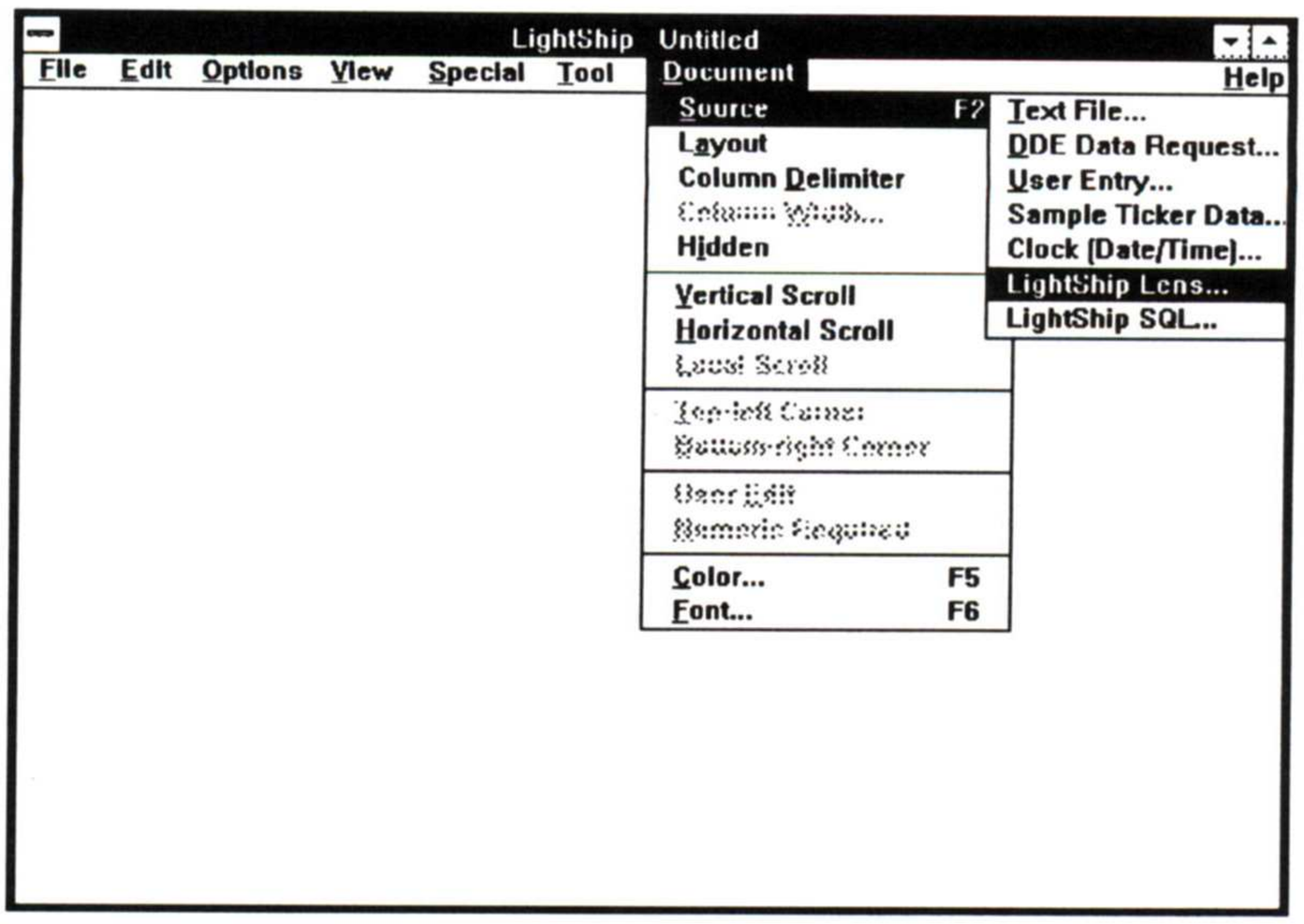

Figure 2-3 Choosing LightShip Lens as a Document Object Source

The Lens window appears.

Figure 2-4 The LightShip Lens Window

Specifying the Database Source

To create a Lens display, you must first select a database source and load specific data that you need for this application and possibly other applications.

Selecting the Database Source

❑ **To select the database source:**

1. Choose File New.

A dialog box appears.

dBASE File, which is the Database Type you want, is selected by default.

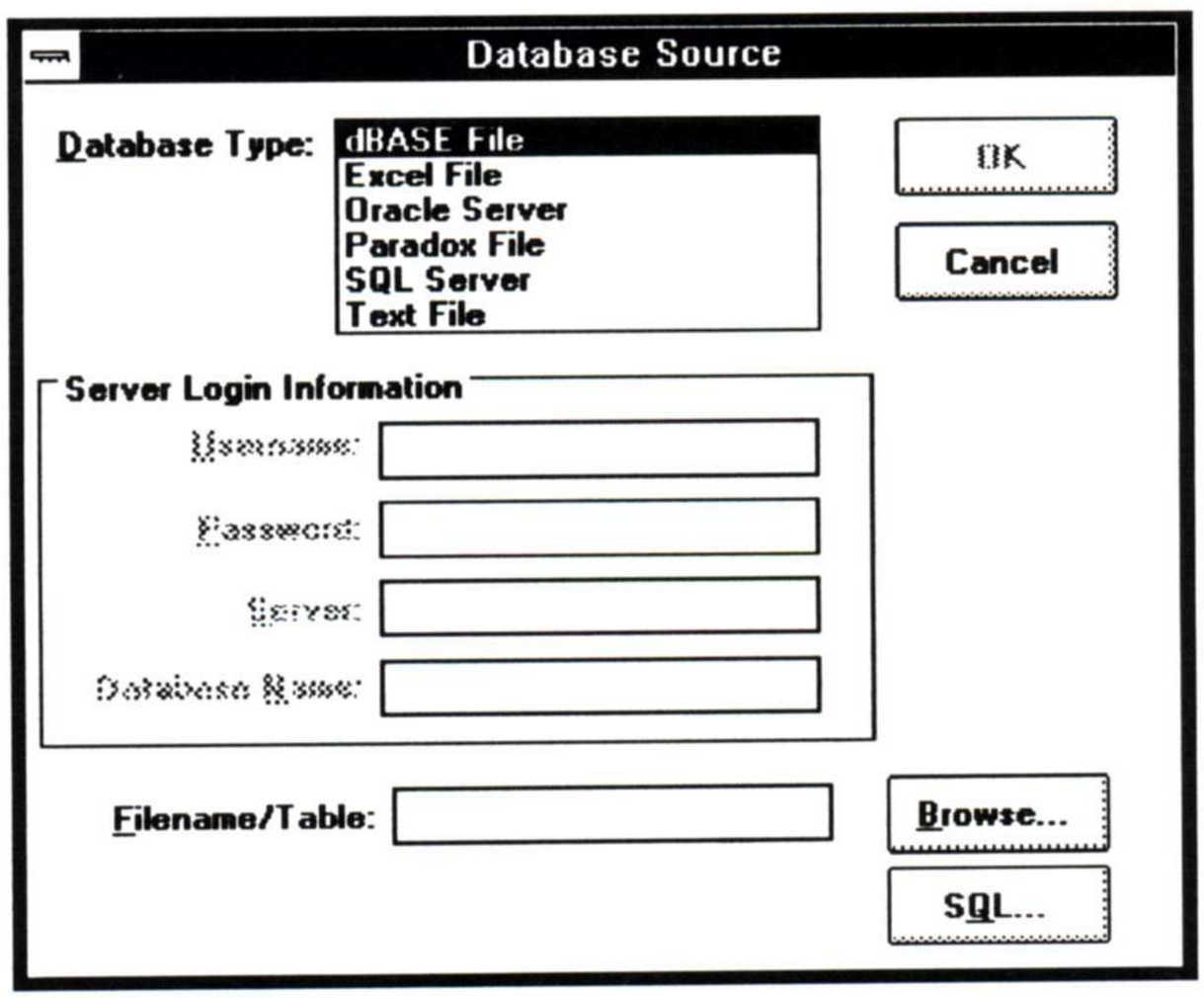

Figure 2-5 Selecting the Database Type

Lens allows you to select a database source from a database table or file or an SQL Select statement. In this case, you are selecting a file, so you can use the Browse button to search through your directories.

2. Choose Browse.

   A dialog box appears.

3. Select the LS\DEMO\DATA directory.

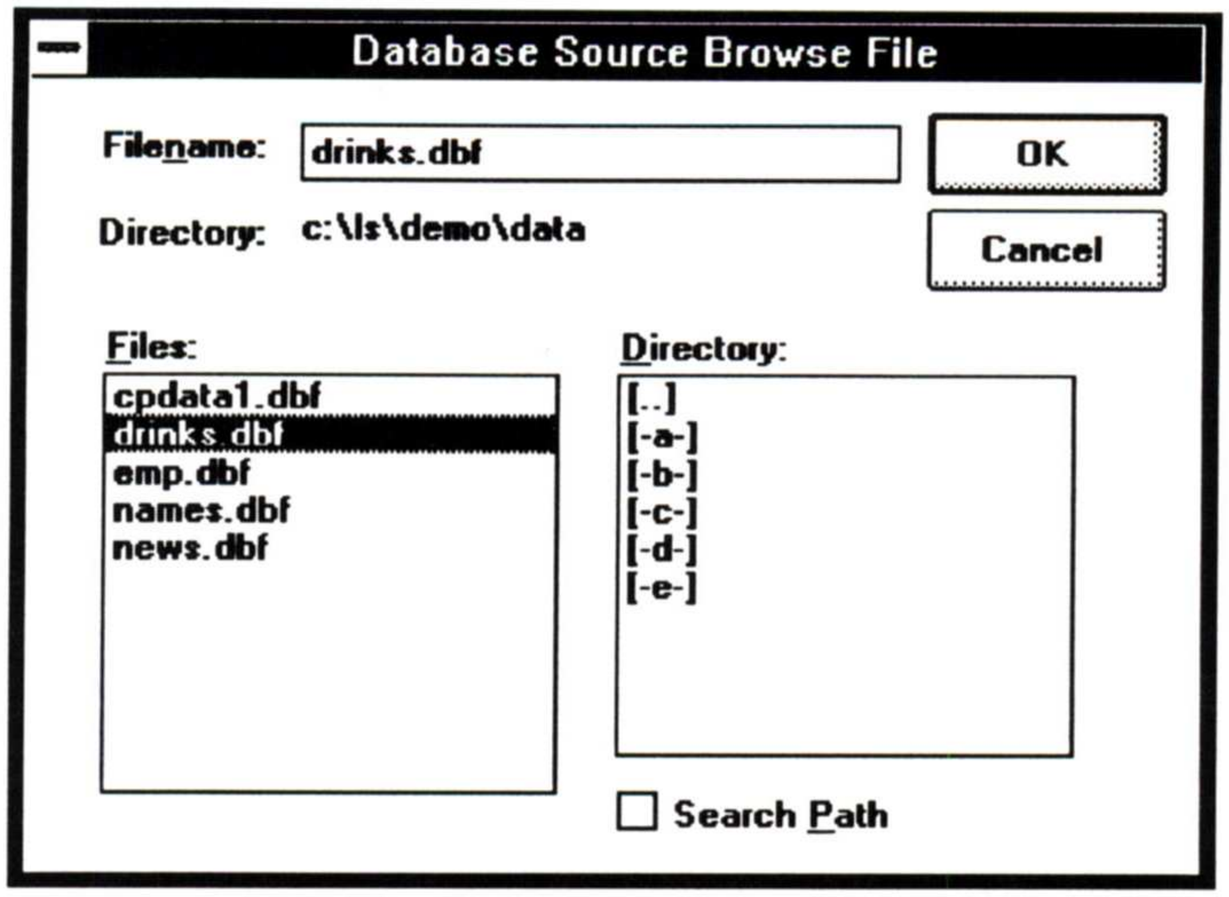

Figure 2-6 Using Browse to Select a Database Source File

4. Select DRINKS.DBF from the Files list box.

☞ **Note**

The Search Path option allows Lens to search for the file in directories other than the current one, using the LightShip and DOS paths. Turn on Search Path whenever the path to the database source may change. For example, you might develop an application in one directory but plan to move it or the database source. For more information, see the *LightShip User's Guide*.

5. Choose OK.

   The Database Source dialog box appears again and the Filename/Table text box contains the filename you selected, as shown in Figure 2-7.

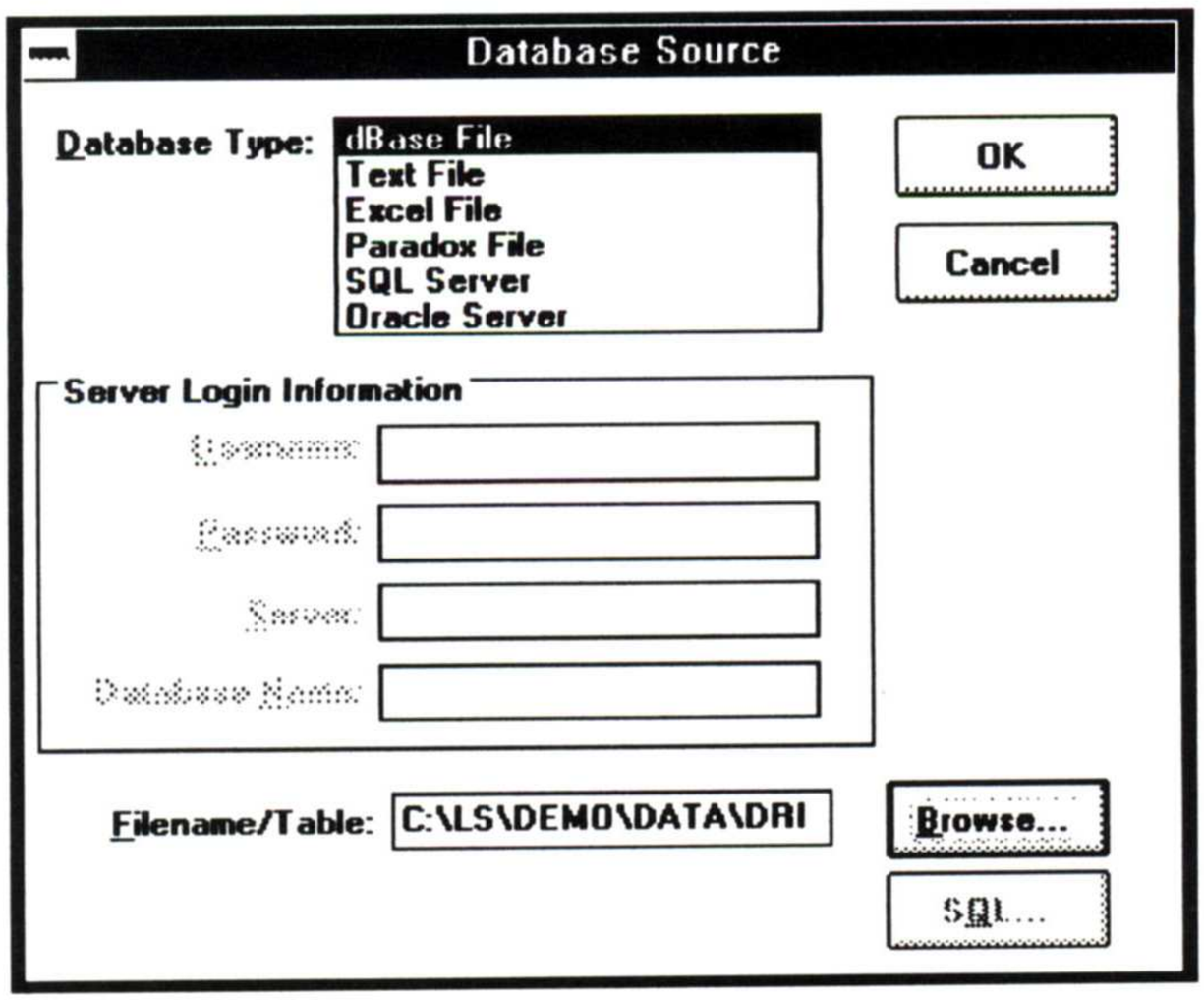

Figure 2-7 Selecting the Database Source

6. Choose OK.

The prompt "Load the database from the source?" appears.

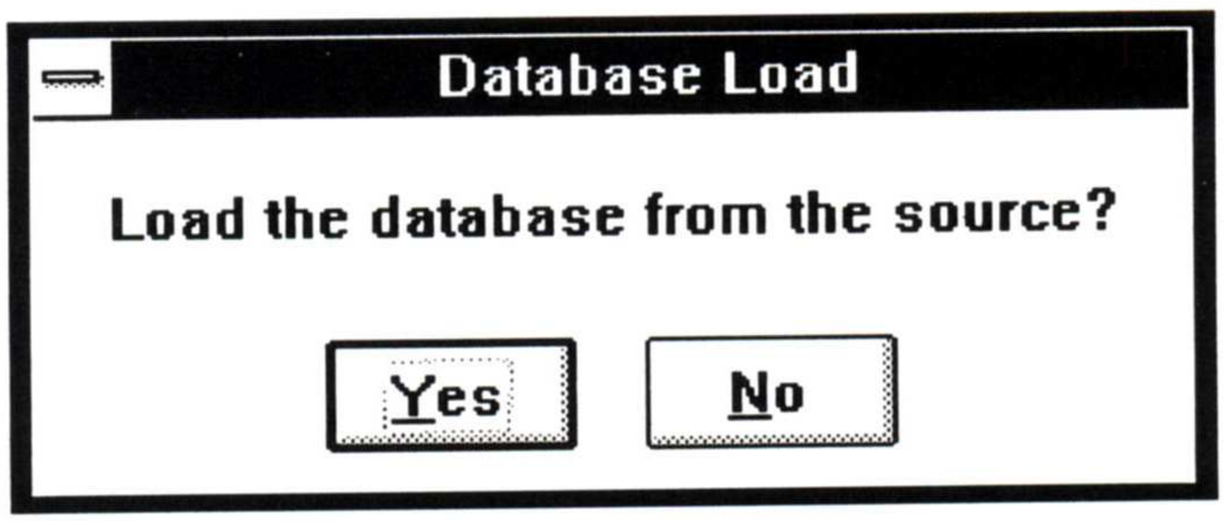

Figure 2-8 Database Load Prompt

This prompt appears as soon as Lens has enough information to load. You do *not* want to answer "Yes" yet because Lens would load all the DRINKS.DBF data and you want to load only a portion of it.

7. Choose No.

A message appears. Disregard it for now because you are still limiting the data cache.

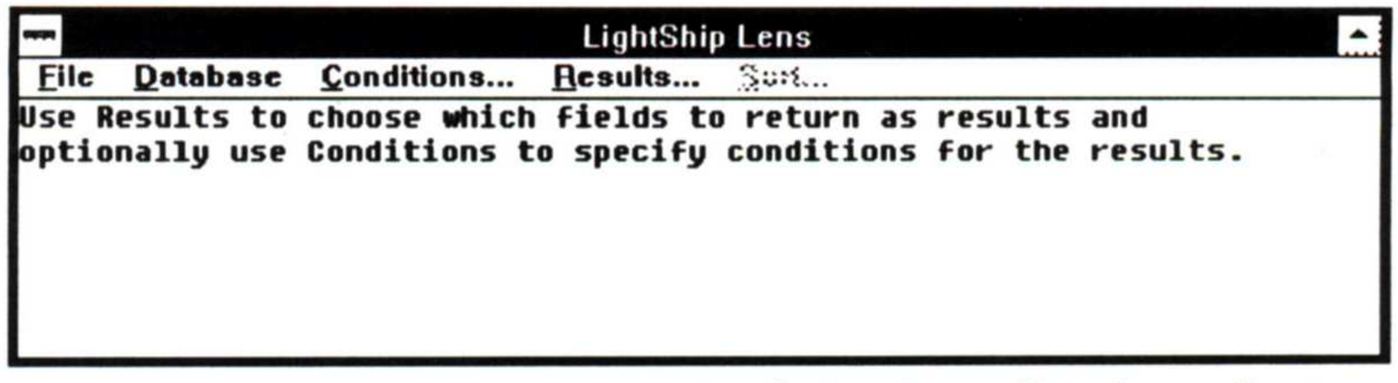

Figure 2-9 The Lens Window after Selecting a Database Source

Choosing the Conditions to Load Data

These steps load only the data from the Northeast region.

❑ **To load conditional data:**

1. Choose Database Load Conditions.

   A dialog box appears.

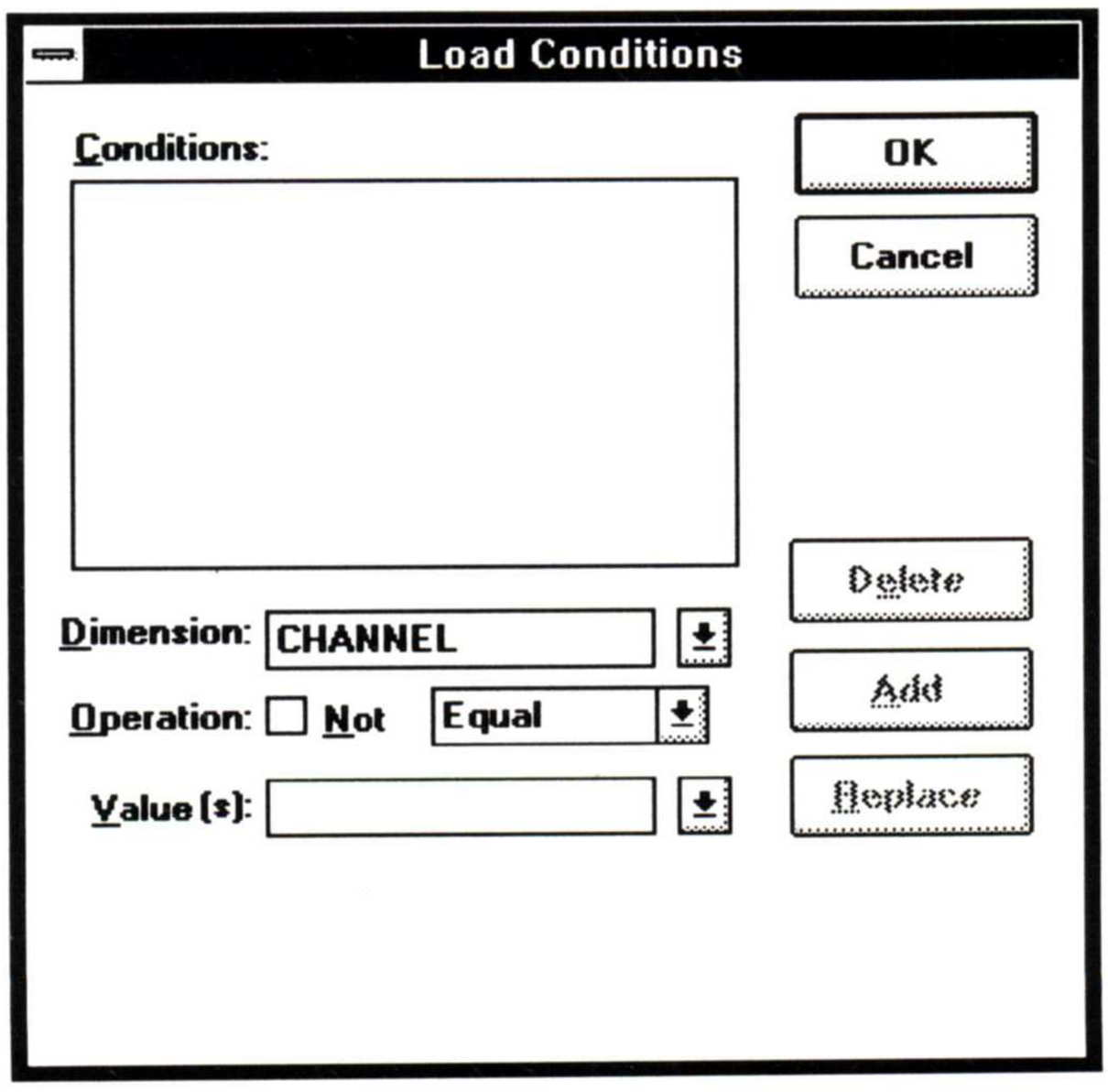

Figure 2-10 Selecting Load Conditions

2. Select the Dimension drop-down list.

   It lists all the dimension names that exist in the current database source.

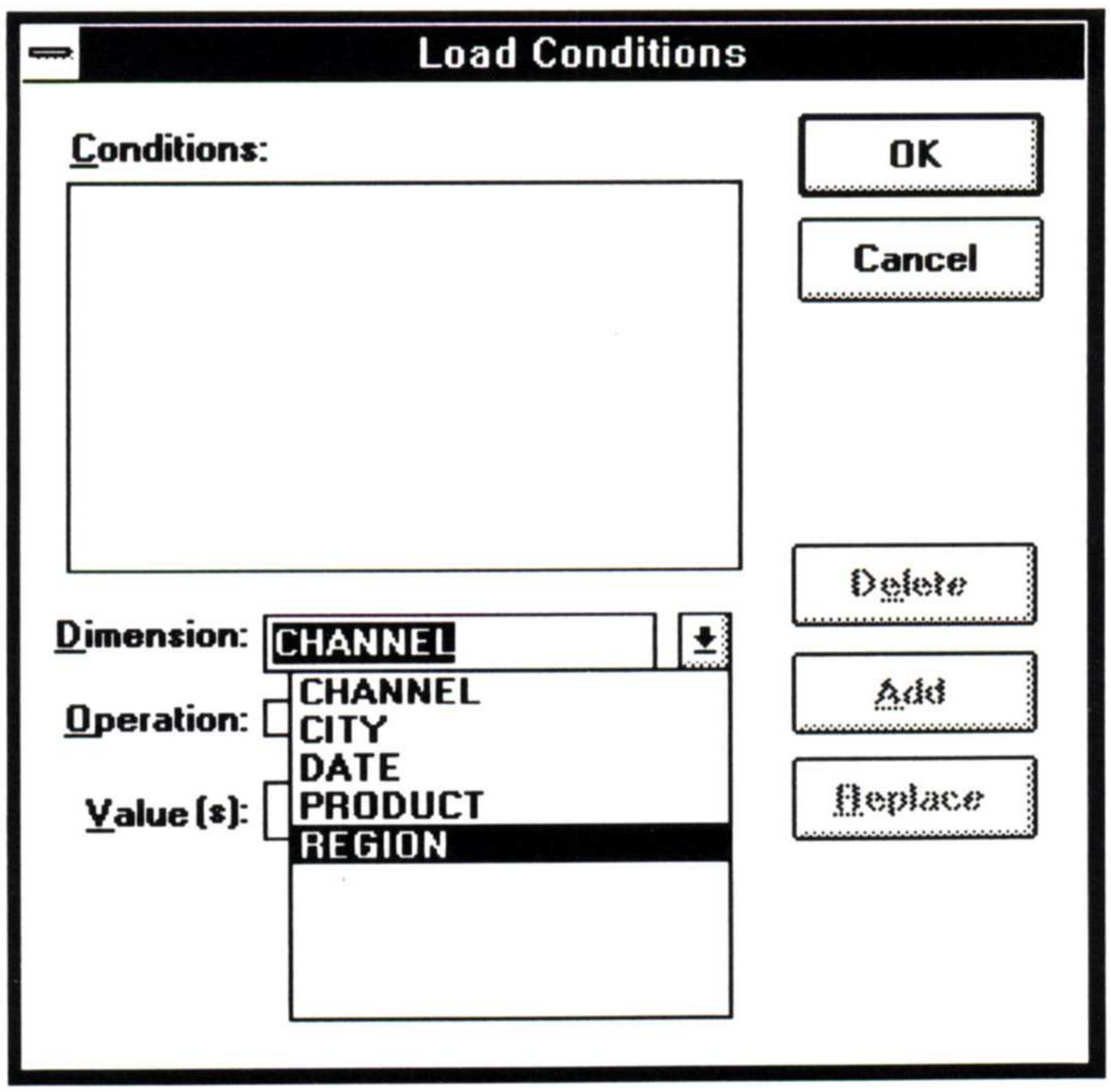

Figure 2-11 Selecting a Dimension

3. Select REGION.

   Since the default operation, Equal, is the one you want, leave the Operation box as it is.

4. Select the Value(s) text box and type:

   `Northeast`

   The dialog box should look like Figure 2-12.

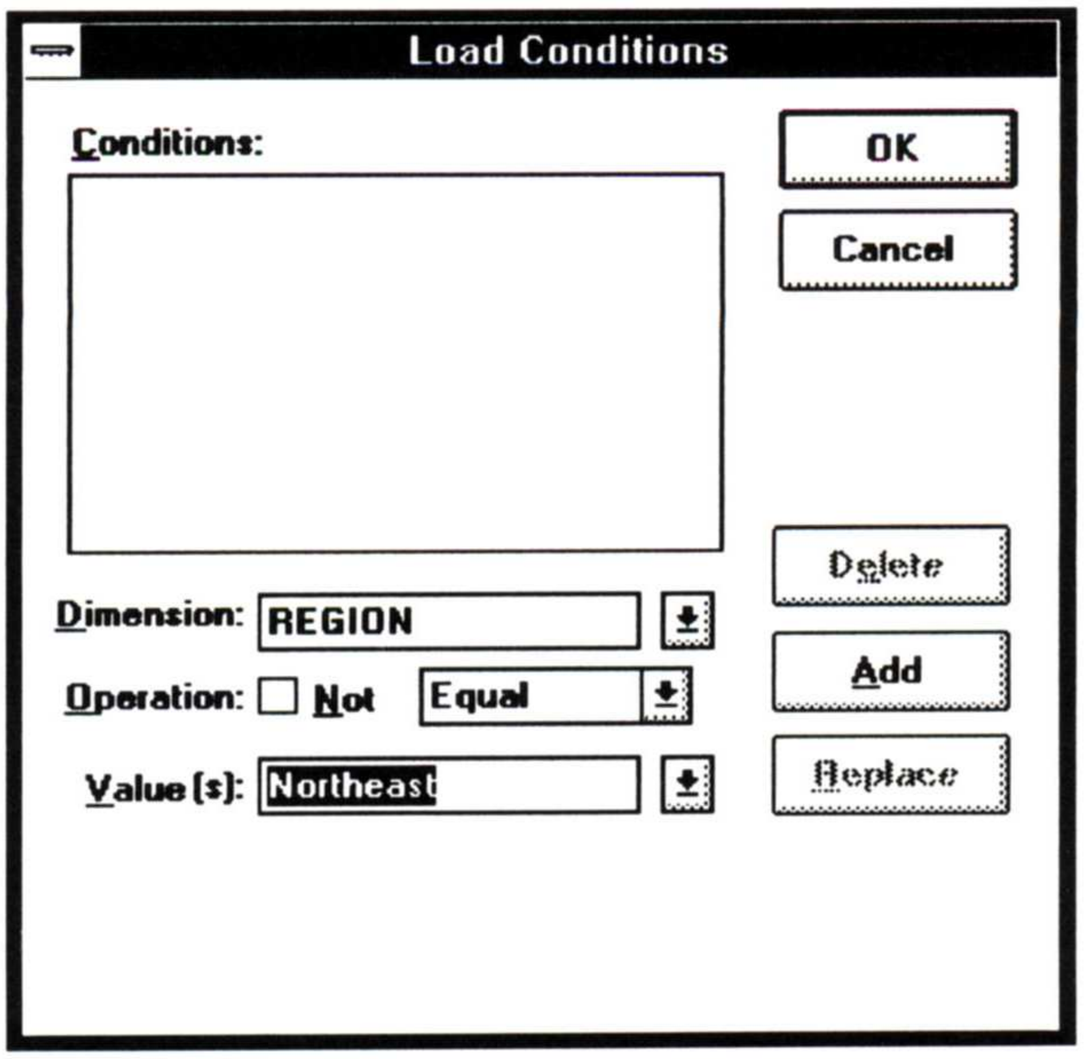

Figure 2-12 Selecting the Dimension's Value

5. Choose Add.

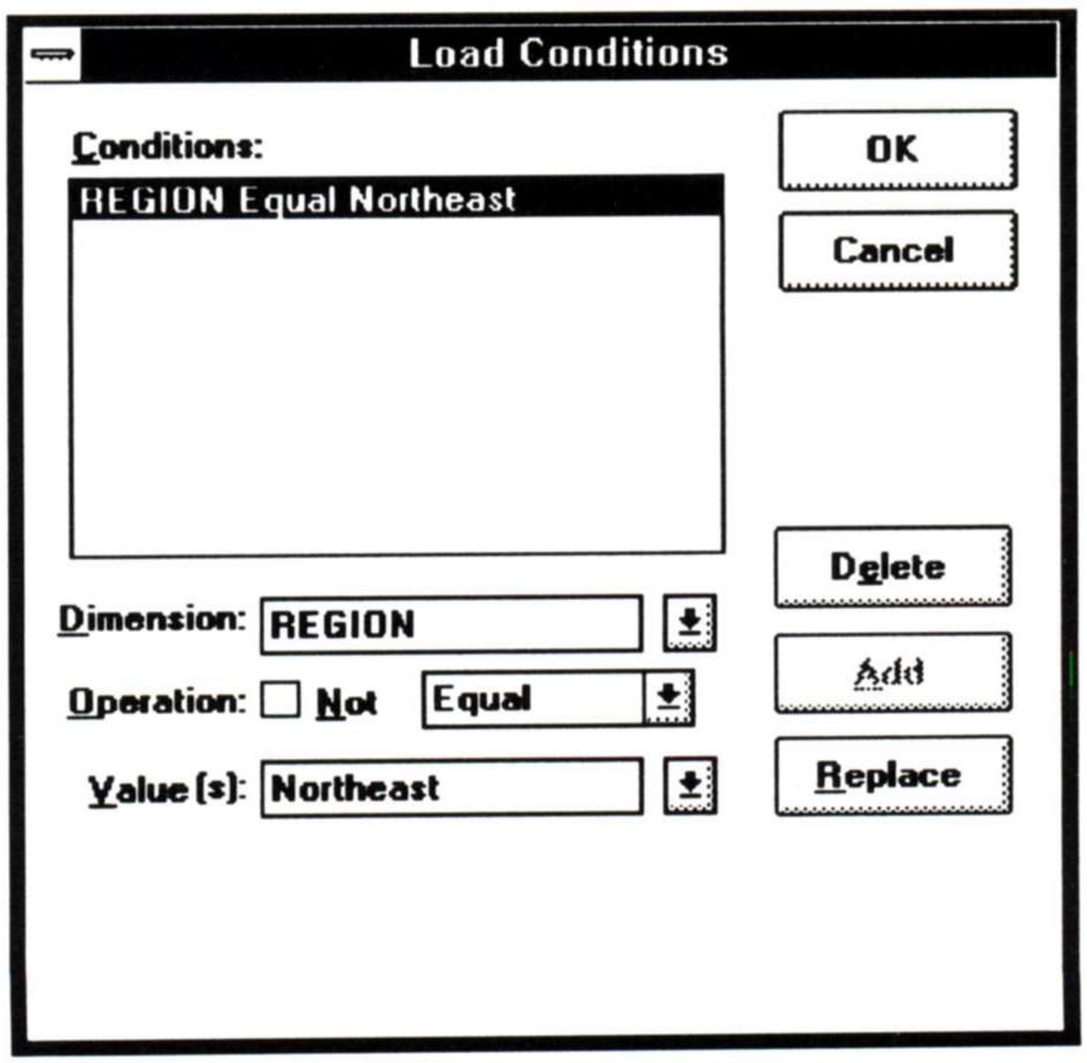

Figure 2-13 Adding the Condition

6. Choose OK.

   The prompt "Load the database from the source?" appears. First, you want to specify the fields to load.

7. Choose No.

Excluding Fields From Being Loaded

Excluding unnecessary fields reduces the size of the data cache and makes the display easier to use. However, extra fields do not affect the display. Lens ignores any fields that you do not explicitly include in the display selections.

❑ **To exclude certain fields from being loaded:**

1. Choose Database Load Fields.

   A dialog box appears.

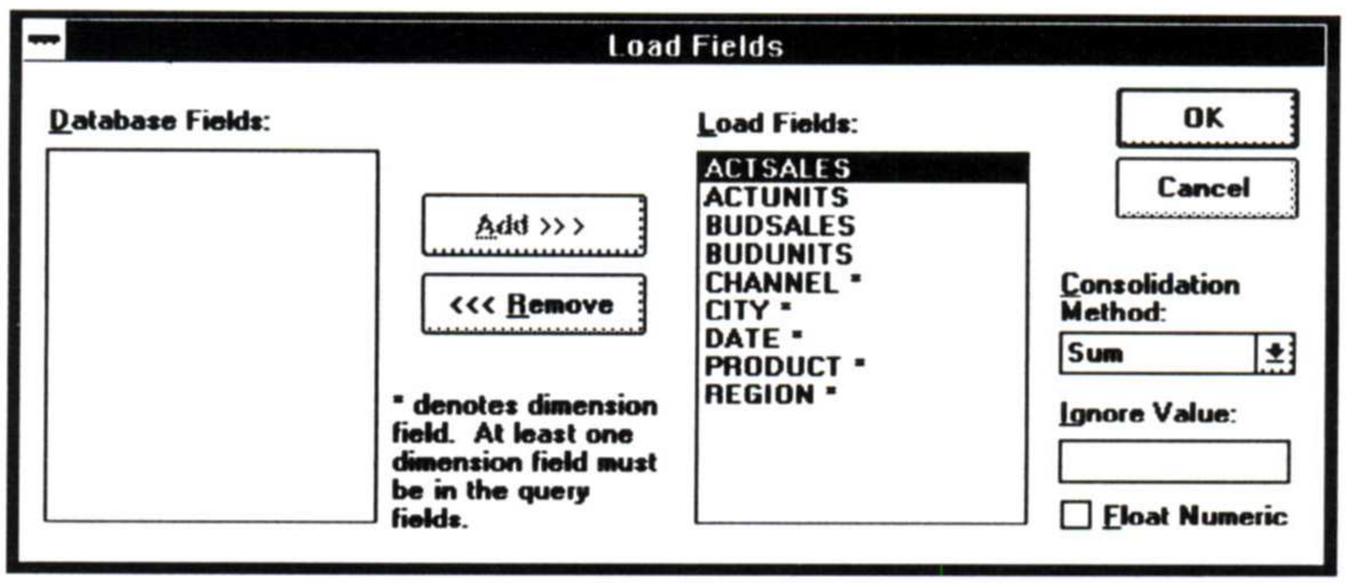

Figure 2-14 Excluding Fields From Being Loaded

   Lens assumes that you want to load all the fields from DRINKS.DBF, so all the fields appear in the Load Fields list. You do not want to load CHANNEL or CITY.

2. Select CHANNEL from the Load Fields list.

3. Choose Remove.

   CHANNEL moves to the Database Fields.

4. Select CITY from the Load Fields list.

5. Choose Remove.

   The dialog box should look like Figure 2-15.

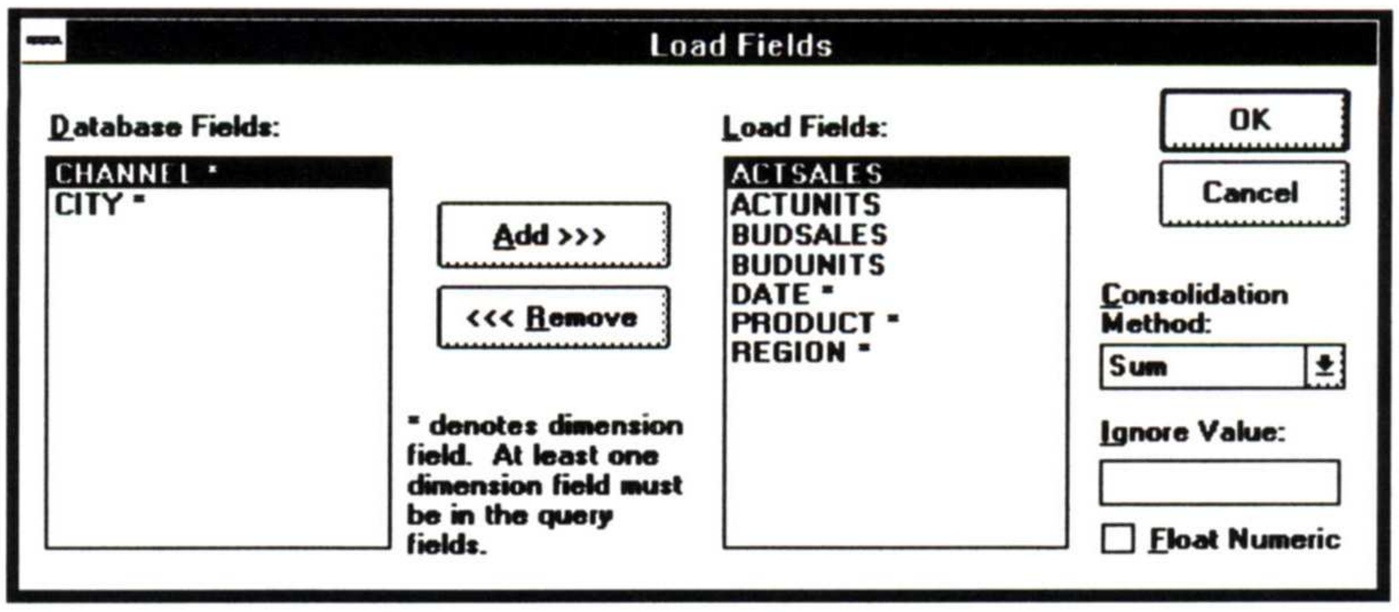

Figure 2-15 Viewing the Fields to be Loaded

Viewing Consolidation Methods

By default, Lens consolidates numeric field values using the Sum consolidation method. You can select a different method for numeric fields (the fields without asterisks).

❑ **To look at the other possible consolidation methods:**

1. Choose ACTSALES in the Load Fields List.

   The Consolidation Method box shows the SUM method.

2. Select the Consolidation Method drop-down list to view all the methods.

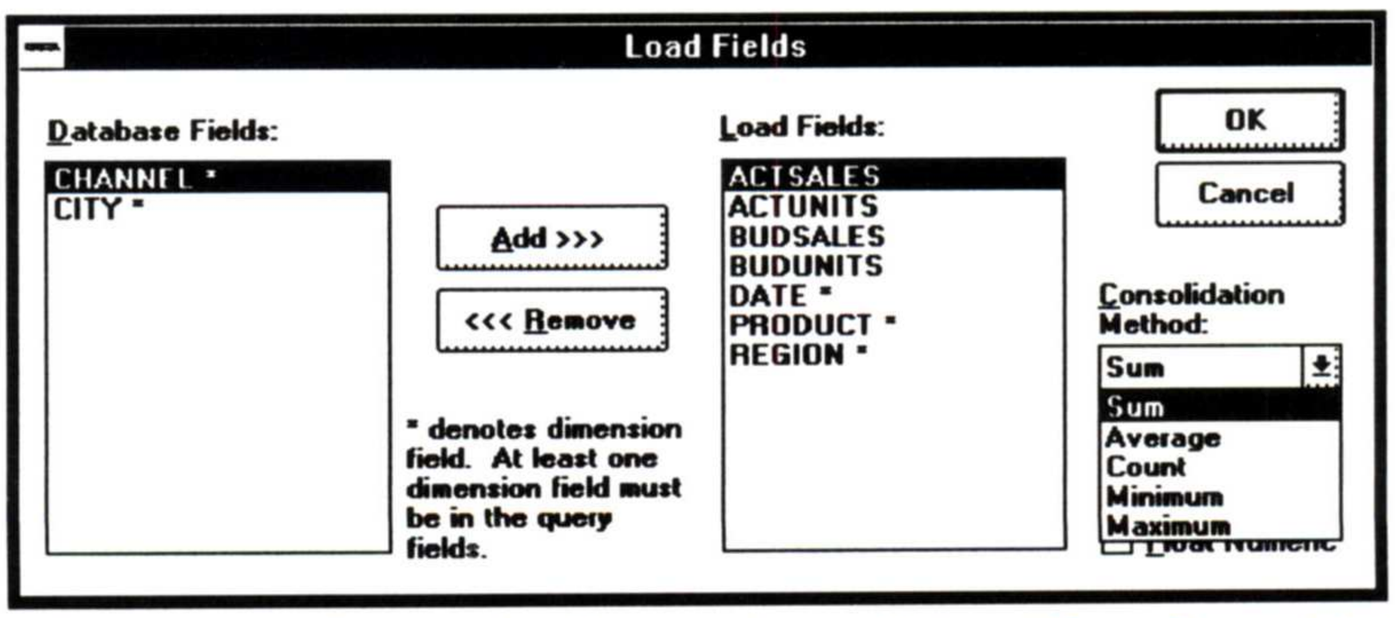

Figure 2-16 Viewing the Consolidation Method Drop-Down List

You want to use Sum, which is already selected as the consolidation method.

3. Choose OK.

   Lens displays the prompt "Load the database from the source?".

Loading the Database Source

❑ **To load the data:**

At the prompt to load the database, choose Yes.

You see the message that "The Database is being loaded..." as Lens counts the number of records and creates the data cache.

Saving the Data Cache to an LSC File

Before you create the display, you will save the data cache to an LSC file. This automatically selects it as the source for the document object.

❑ **To save the data cache to an LSC file:**

1. Choose File Save As.

   A dialog box appears, and lists the LSC files in the current directory, as shown in Figure 2-17.

2. Select the LS\TUTORIAL directory.

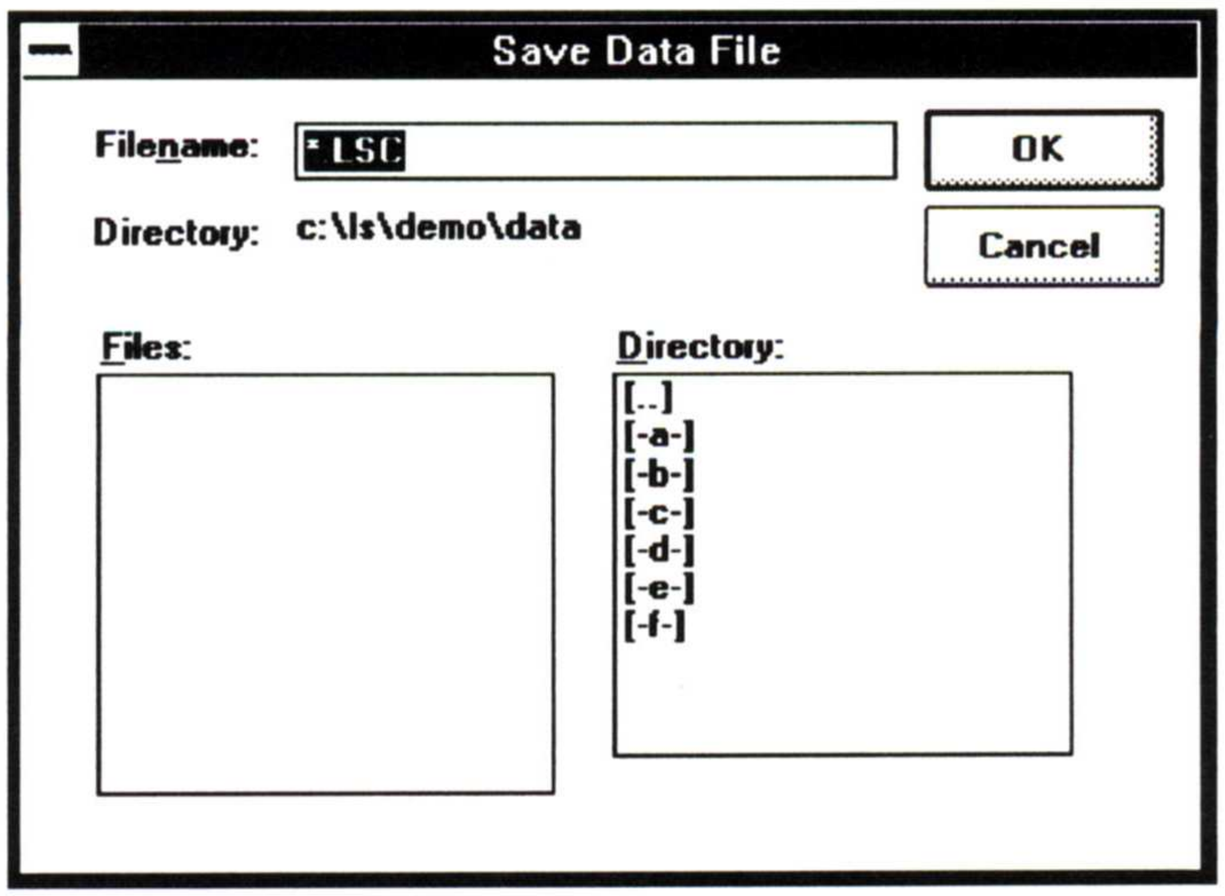

Figure 2-17 Saving Data to an LSC File

3. In the Filename text box, type:

   example.lsc

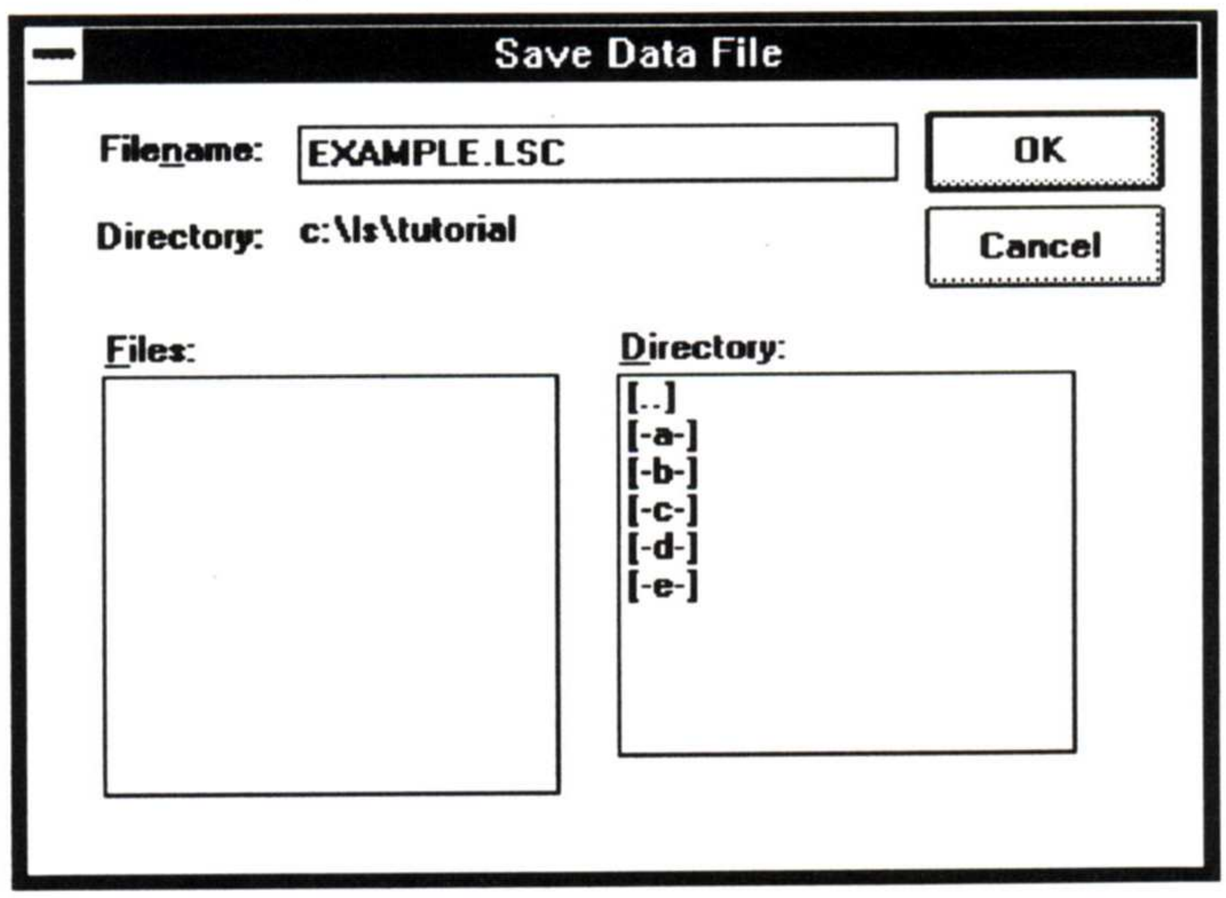

Figure 2-18 Specifying the LSC Filename

4. Choose OK.

   The Lens display is now saved as an LSC file.

Creating the Display

The LSC file contains many dimensions and fields that you can use for various applications. For this application, you want to display only some of that data.

In this section you will:

- Select the fields, the dimension directions, and column headings.
- Enter a condition to display data for selected years.

Choosing the Fields to Display

❑ **To specify the fields to display:**

1. Choose Results.

   A dialog box appears.

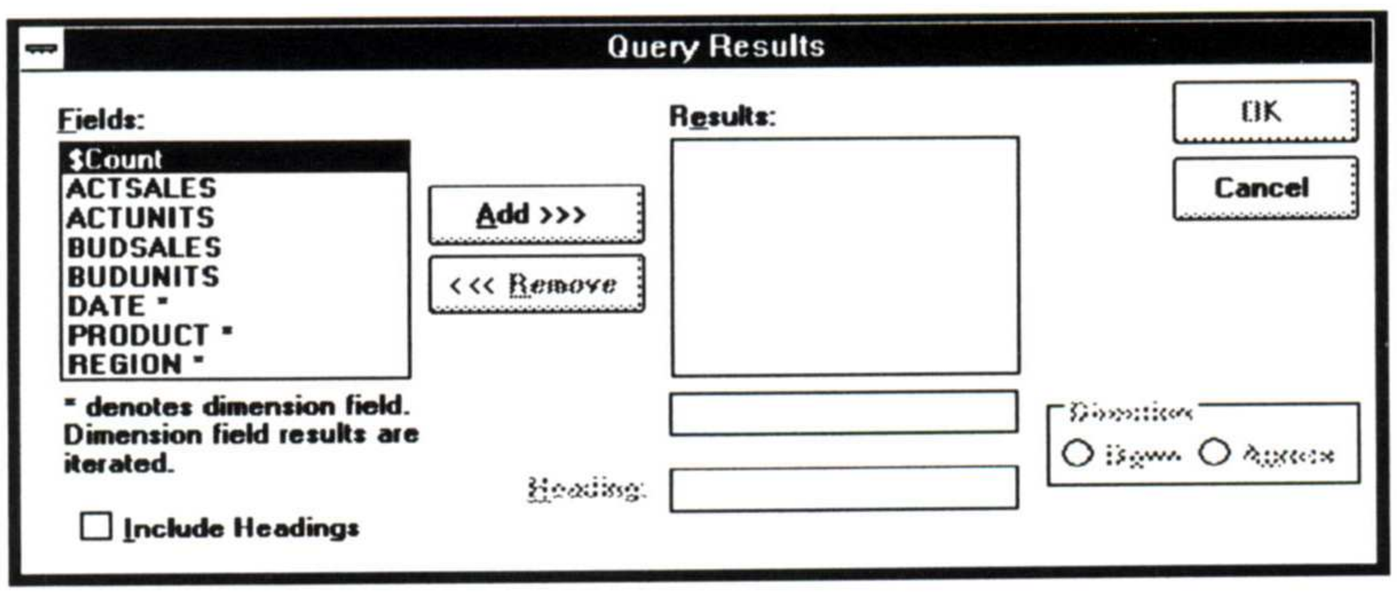

Figure 2-19 Specifying Fields to Display

The Fields list contains all the fields in the data cache. You select the fields to display by adding them to the Results list, which by default contains no fields.

The order of fields in the Results list decides the order in which the fields are displayed. The PRODUCT Dimension field must be at the top of the Results list so that its values will appear first.

2. Select PRODUCT from the Fields list box.

3. Choose Add.

☞ **Note** You can also double-click on an item in one list to move it to the other list. This is true throughout Lens.

4. Continue adding the following fields to the Results list in this order:

   - ACTSALES
   - BUDSALES
   - ACTUNITS
   - BUDUNITS
   - DATE

The dialog box should look like Figure 2-20.

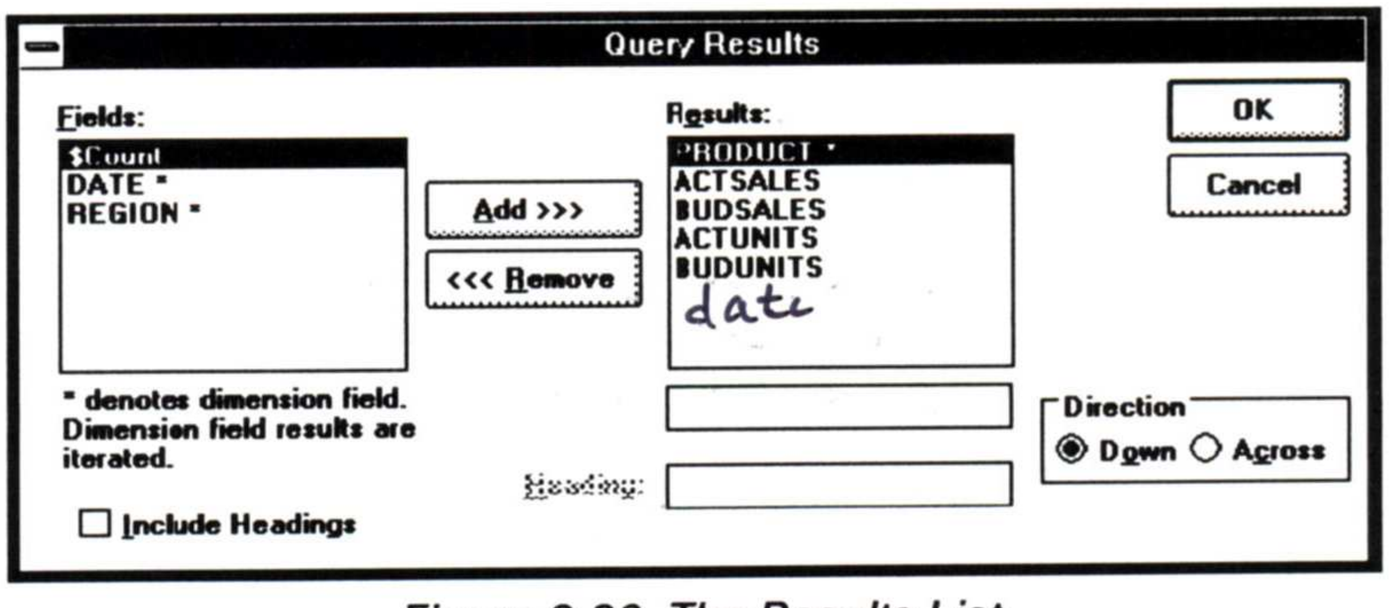

Figure 2-20 The Results List

Specifying the Dimension Direction

Dimension fields (marked with asterisks) determine how Lens arranges data in the display. You can set the direction of each dimension field to Across or Down. In this case, you want PRODUCT to display downward.

❑ **To view the dimension's direction:**

1. Select PRODUCT in the Results list.

   The default direction is Down, so you can leave it as is.

2. Choose OK.

The results are displayed in the Lens window.

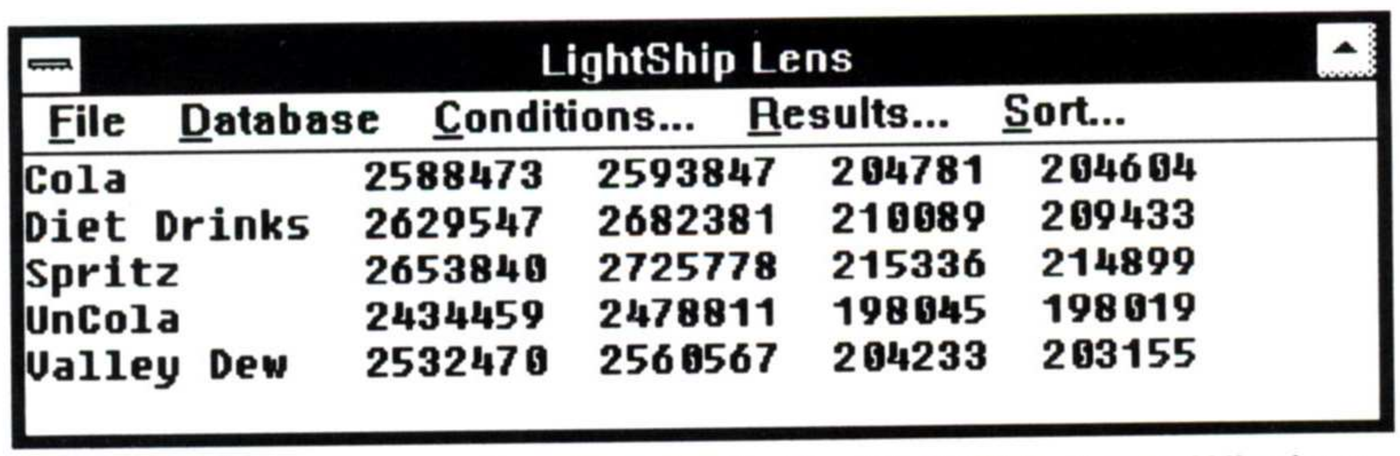

Figure 2-21 Displaying Results in the LightShip Lens Window

Lens always displays the most current selections whenever you change information using the Results or Conditions command.

Displaying Conditional Data by Variable Reference

To display data in LightShip based on the currently selected year, you will enter a condition based on a variable called *date*.

Eventually, you will create a hotspot on the LightShip screen whose action assigns a value to the *date* variable; the document object data will change accordingly.

❑ **To display data based on a *date* variable:**

1. Choose Conditions.

A dialog box appears, as shown in Figure 2-22.

You want to use the DATE dimension and the Equal operation. Since DATE already appears in the Dimension box and Equal is already selected by default, you can leave them alone.

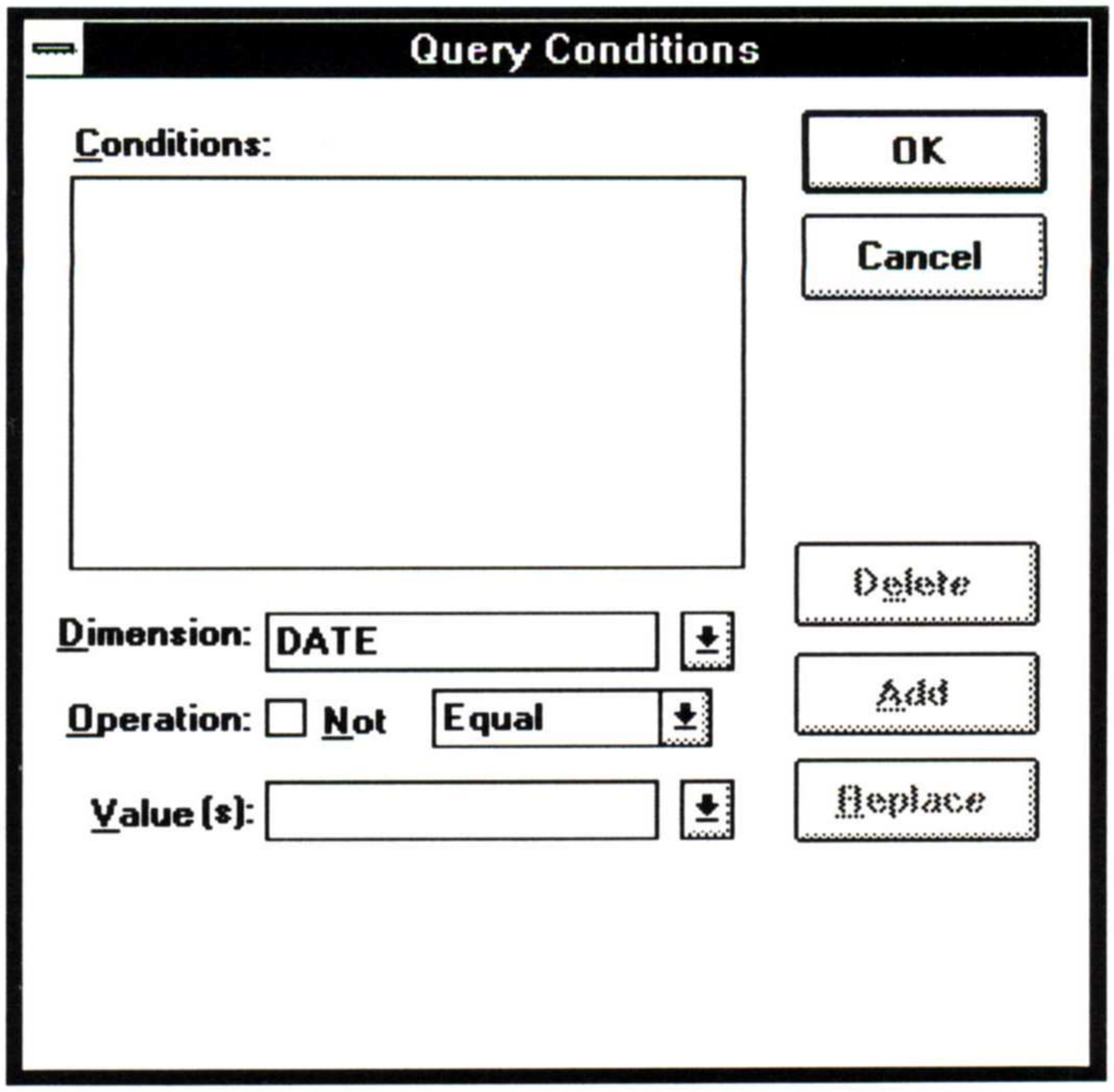

Figure 2-22 Selecting Query Conditions

☞ **Note** Lens stores dates in the format CCYYMMDD.

In conditions click on op box - select "like"

2. Select the Value(s) text box and type:

missing

```
@(date)
```

☞ **Note** You can type a LightShip variable reference in any text box large enough to contain four characters.

3. Choose Add.

   The dialog box should look like Figure 2-23.

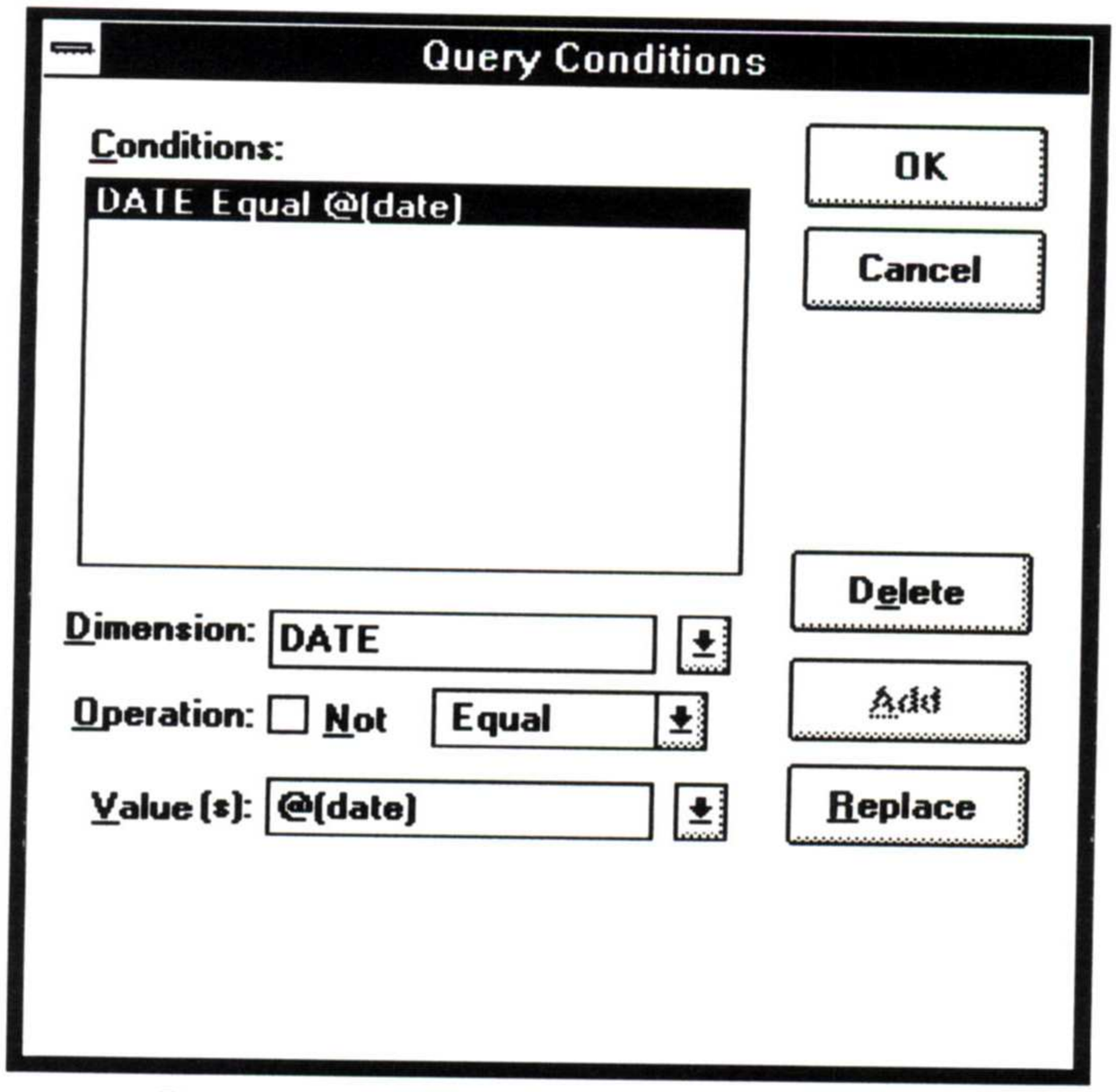

Figure 2-23 Specifying a Query Condition

4. Choose OK.

The Lens display should look like Figure 2-24. **`DATE Equal @(date)`** displays the DATE field records that are equal to the value you assign to *date*. Because the *date* variable currently has a null value, Lens ignores the query condition for now. The display continues to show a consolidated view of *all* the data loaded into the data cache, that is, all the data for the Northeast region.

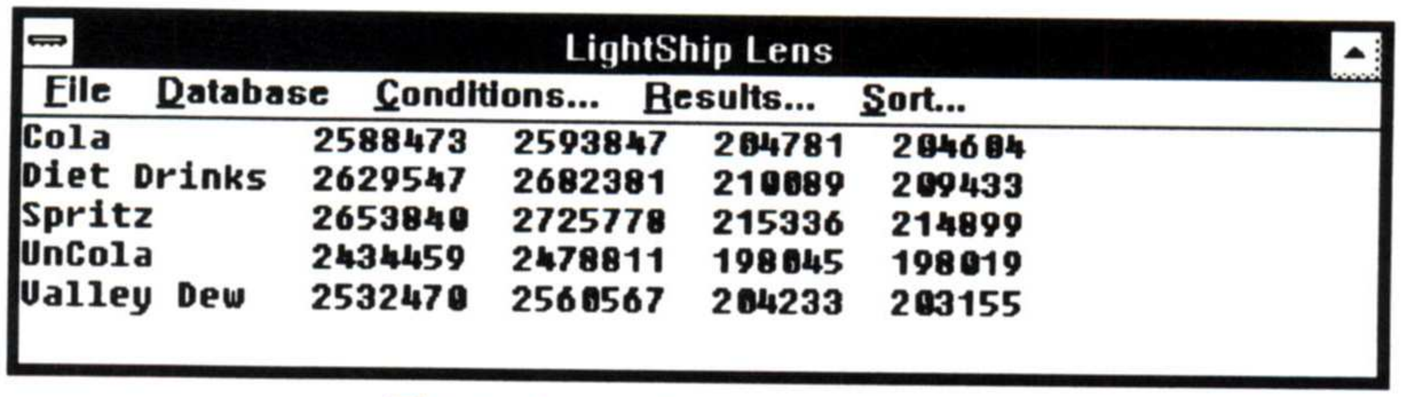

Figure 2-24 The Lens Window

Adding Column Headings

Before you leave Lens to display the data in a document object, you can add column headings.

❑ **To include headings:**

1. Choose Results.

   A dialog box appears, as shown in Figure 2-25.

2. Turn on Include Headings.

   This displays column headings for the fields. By default, it uses the database source's field names.

❑ **To change the field names to text strings:**

1. Select ACTSALES in the Results list box.

2. Select the Heading text box and type:

   `Actual Sales`

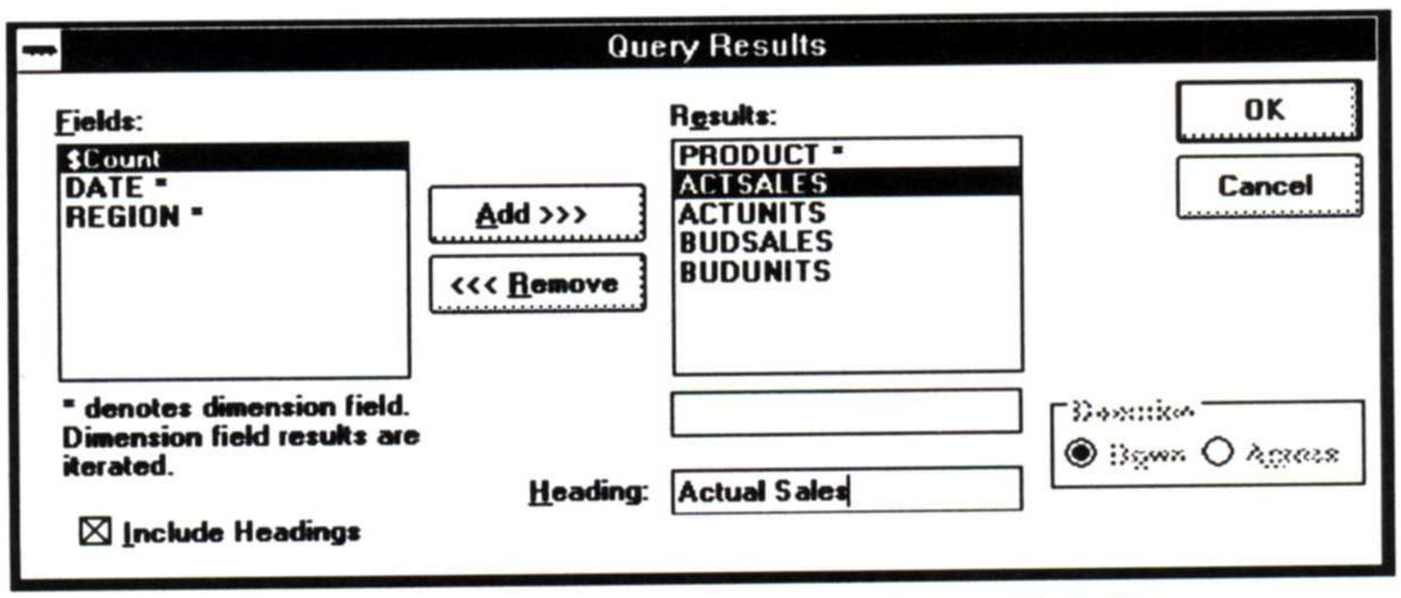

Figure 2-25 Typing the Actual Sales Heading

3. Select BUDSALES.

4. Select the Heading text box and type:

   `Budgeted Sales`

   The dialog box should look like Figure 2-26.

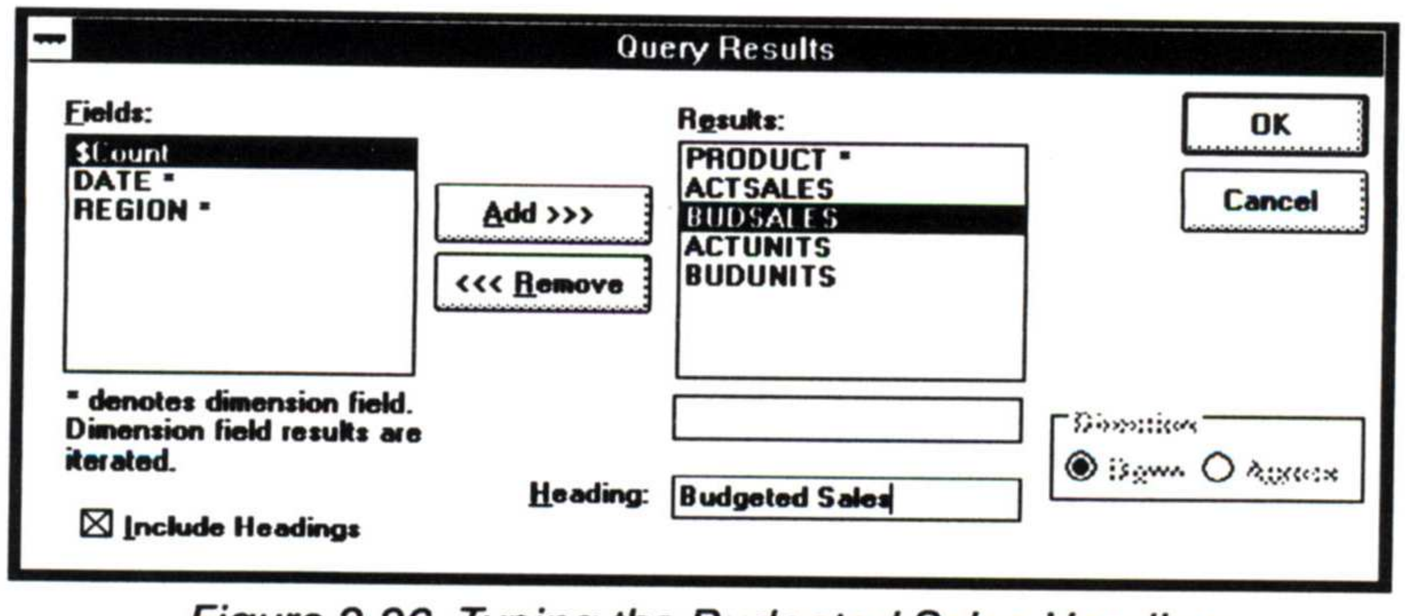

Figure 2-26 Typing the Budgeted Sales Heading

5. Select ACTUNITS.
6. Select the Heading text box and type:

   Actual Units

7. Select BUDUNITS.
8. Select the Heading text box and type:

   Budgeted Units

9. Choose OK.

   The column headings appear above the columns of data.

Figure 2-27 Adding Column Headings

Saving the Display

❑ **To save the Lens display selections:**

Choose File Exit/OK.

This saves the display and returns you to LightShip. The data appears in the document object but some headings are hidden, so you must adjust the default widths.

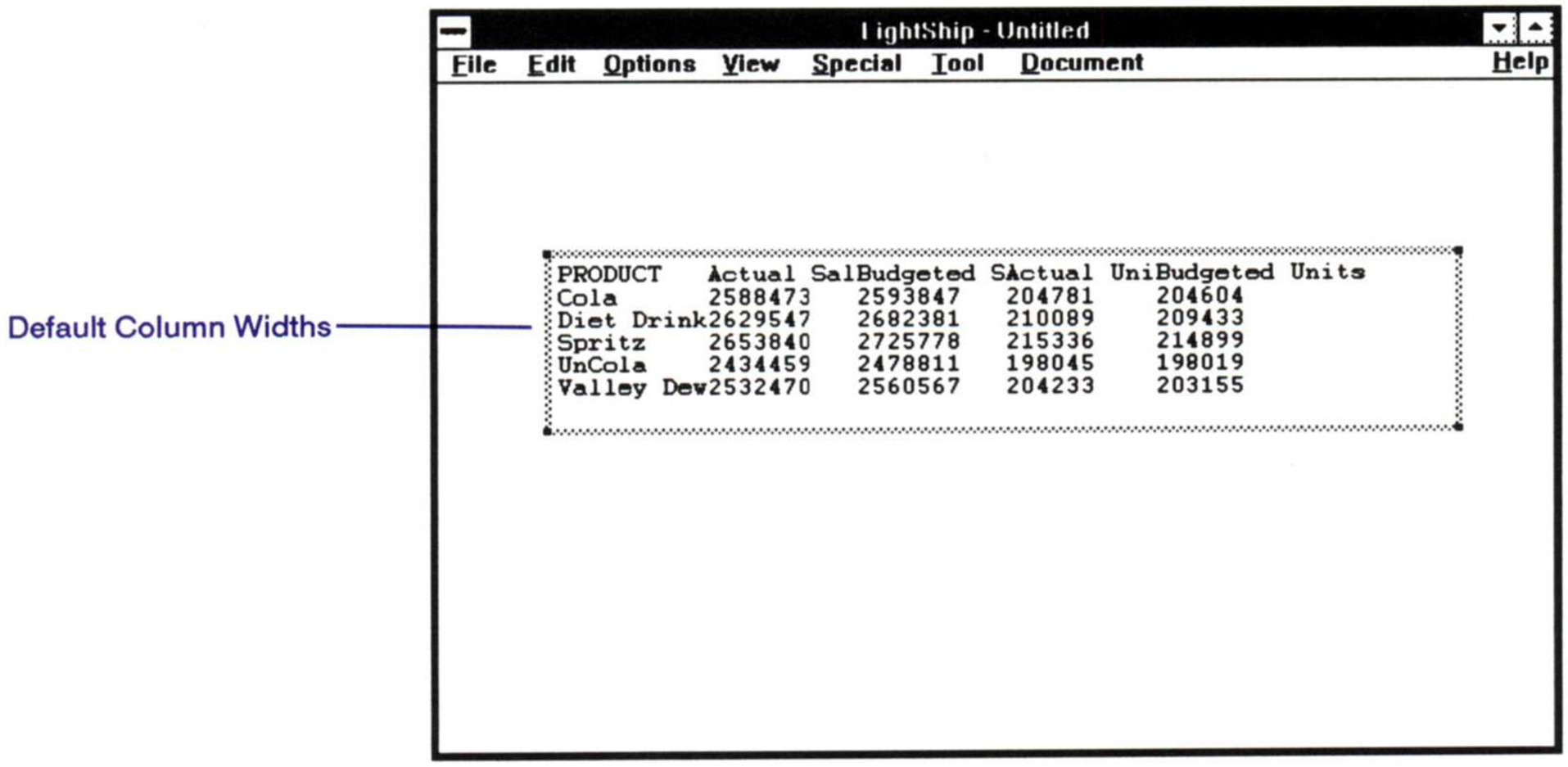

Figure 2-28 Displaying Lens Data in a Document Object

❑ **To adjust the column widths:**

1. Choose Document Column Width.

   A dialog box appears.

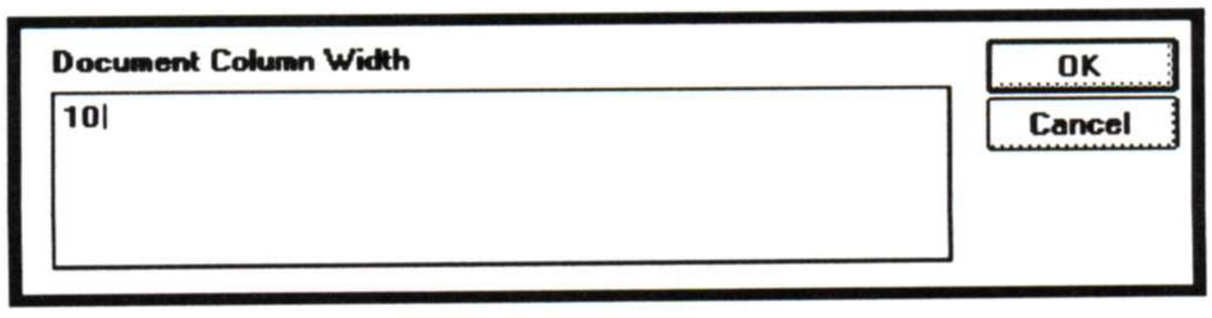

Figure 2-29 Changing the Default Column Widths

2. Type **15** in place of the 10.

3. Choose OK.

   The document object displays the adjusted columns. You might need to resize the document object to display all the data.

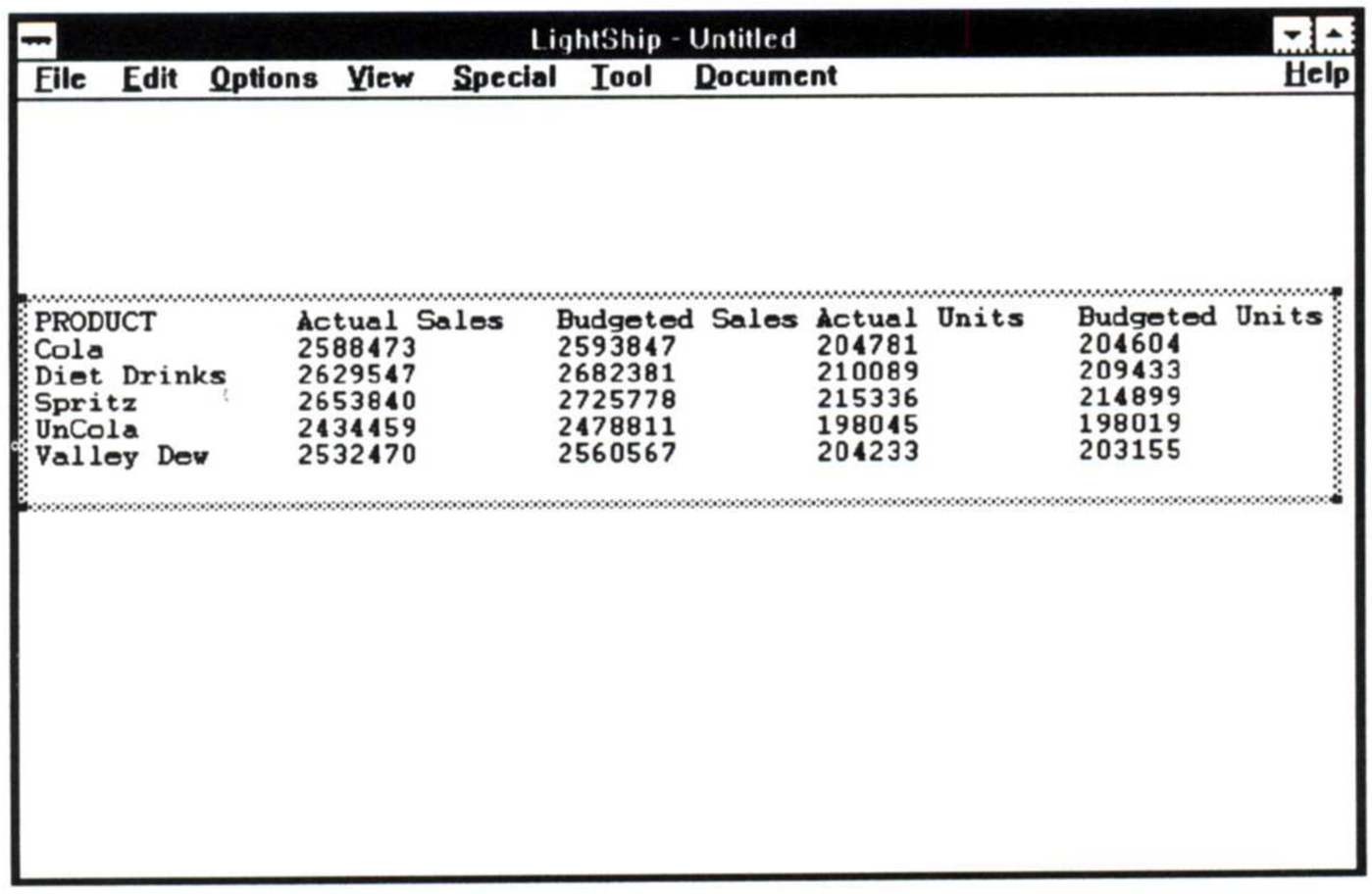

Figure 2-30 Displaying Adjusted Column Widths

❑ To save this LightShip screen:

1. Choose File Save As in LightShip.

   A dialog box appears so that you can specify the filename.

2. In the Filename text box, type:

   `example.lsf`

3. Choose OK.

   You can continue to the next exercise or quit from LightShip and return to it later.

Displaying Lens Data Dynamically in LightShip

This exercise shows you how to create hotspots that display the Northeast region's sales figures for 1989 or 1990. To do this, you will assign a value to the variable *date*, which is referenced in the query condition.

❑ **To start:**

1. On the LightShip screen, choose File Open.

   A dialog box appears.

2. Choose the file EXAMPLE.LSF from the LS\TUTORIAL directory.

   The EXAMPLE.LSF file appears.

3. Move the document object to the lower half of the screen.

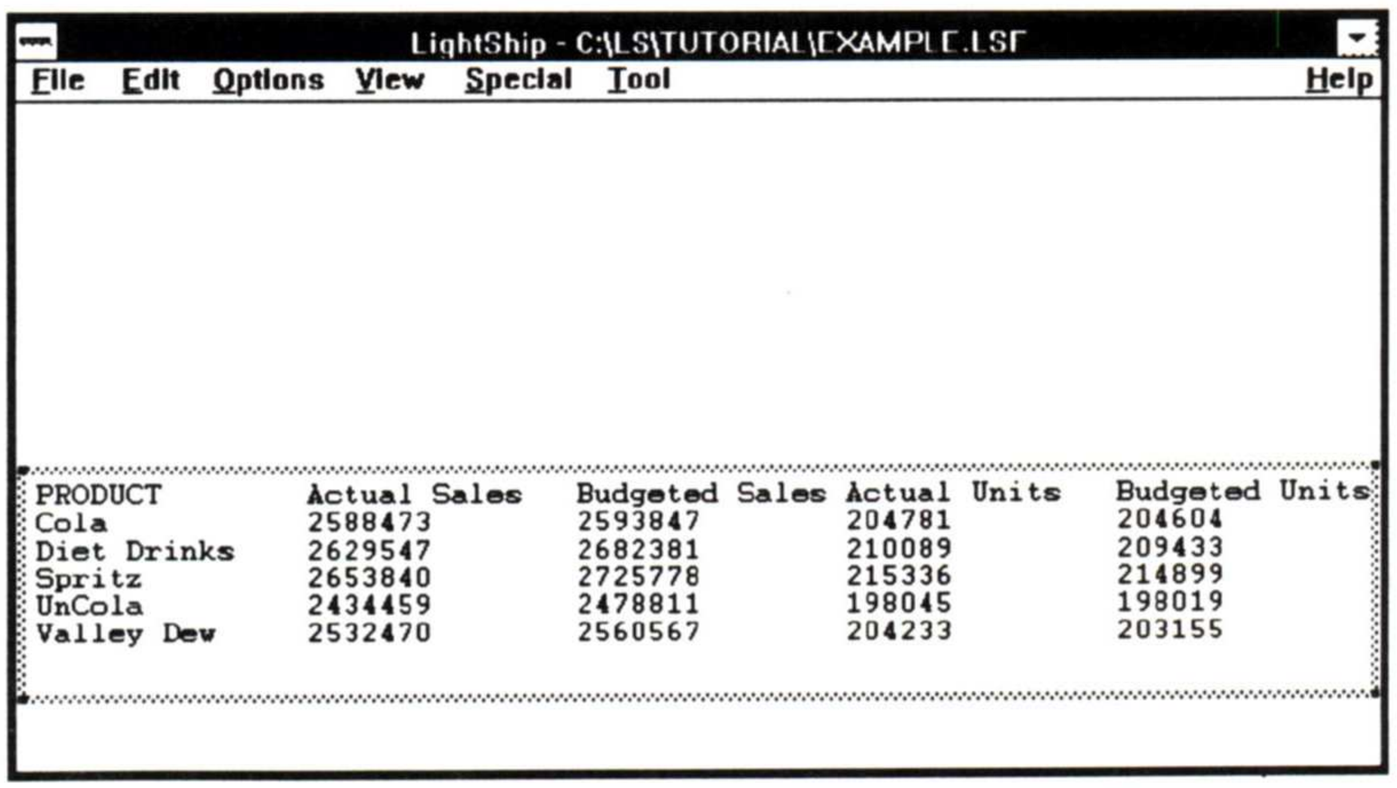

Figure 2-31 Displaying Lens Data in LightShip

Changing the Document Object's Display by the Date

The following steps create the hotspots that will change the value of the *date* variable and thereby change the document object's display.

First, you will initialize a value for *date*.

❑ **To set the *date* variable:**

1. Choose Special Variables.

2. In the Name text box, type:

   `date`

   and press TAB.

3. In the Value text box, type:

   `1989`

   and click anywhere in the Name = Value list box.

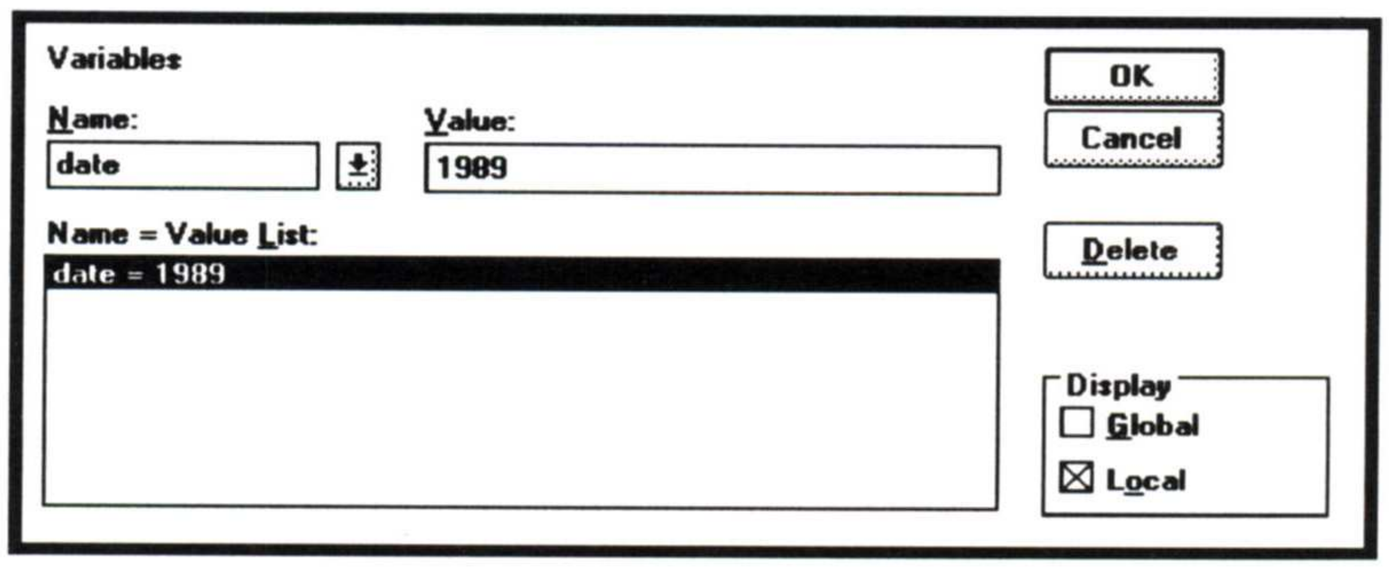

Figure 2-32 Assigning a Value to the Date Variable

4. Choose OK.

The document object should look like Figure 2-33.

Now you can create the hotspots that assign the 1989 and 1990 values to the *date* variable. Selecting either one at the Browse level will automatically update the data based on the current value of *date*.

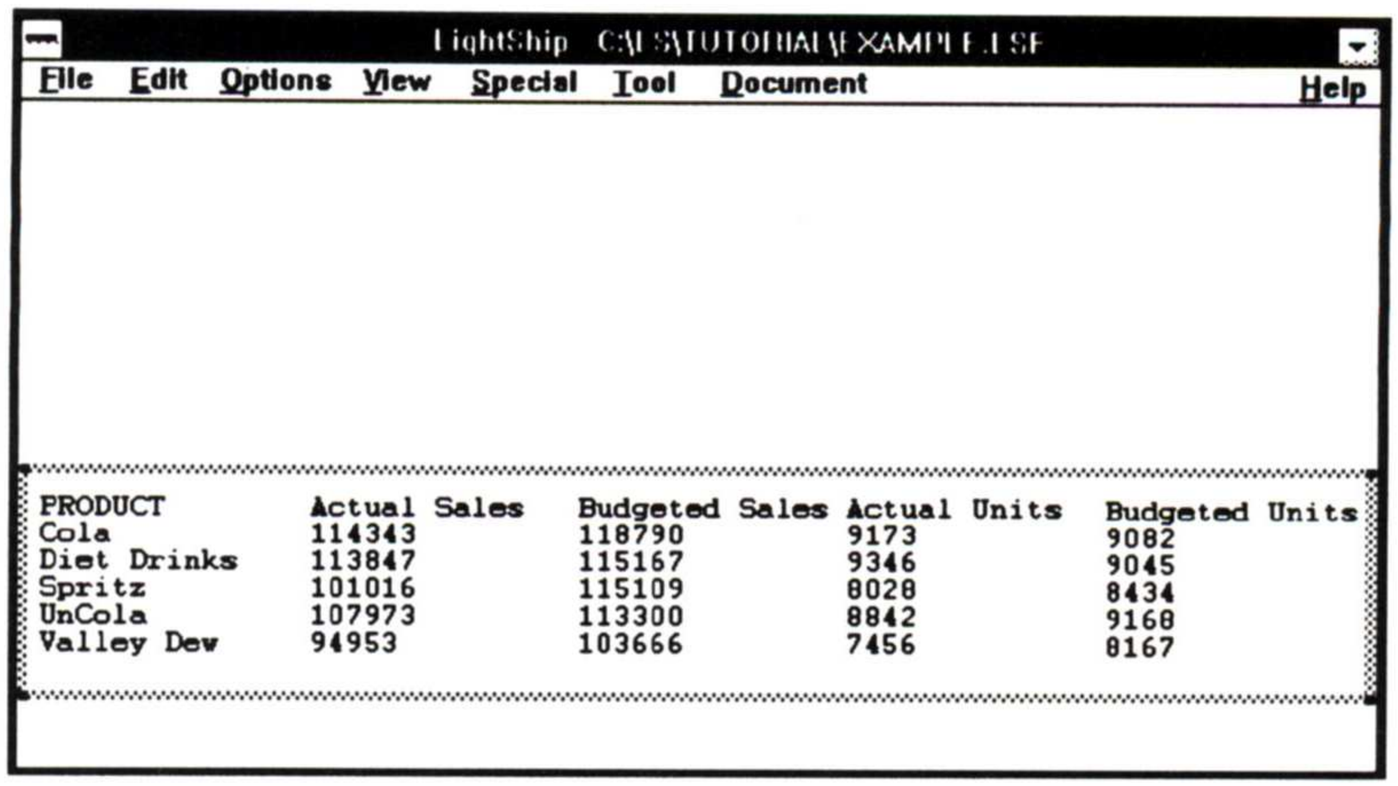

Figure 2-33 Displaying a Document Object with 1989 Data

❑ To create the first hotspot:

1. Choose Tool Hotspot and drag the mouse to create the hotspot object in the area shown in Figure 2-34.

2. Choose Hotspot Highlight Group 1.

3. Choose Hotspot Button Style.

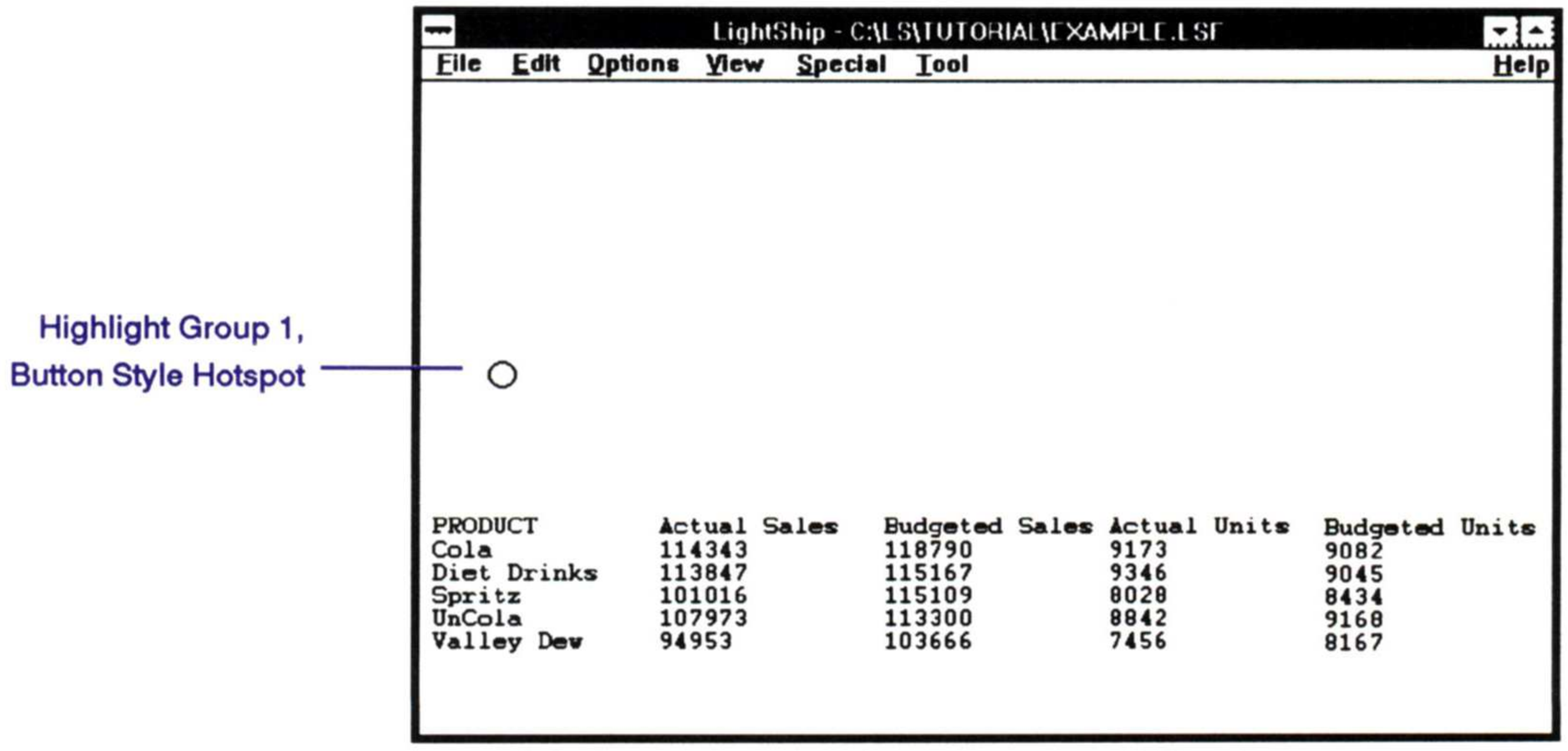

Figure 2-34 Creating a Hotspot

4. Choose Hotspot Actions.

   A dialog box appears.

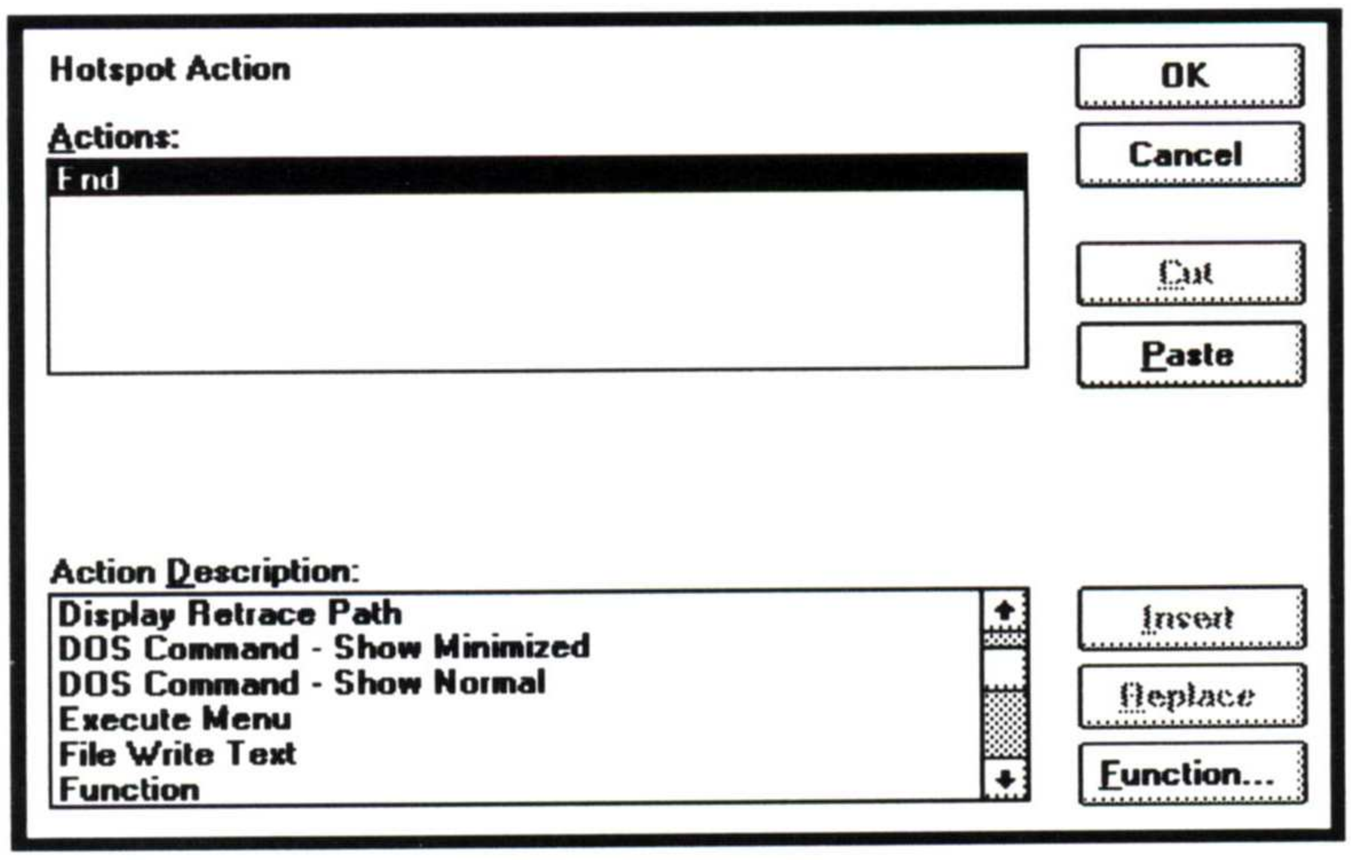

Figure 2-35 Choosing Hotspot Actions

5. Choose the Function button.

   A dialog box appears.

6. Specify the Copy/Test string function:

```
Date := "1989"
```

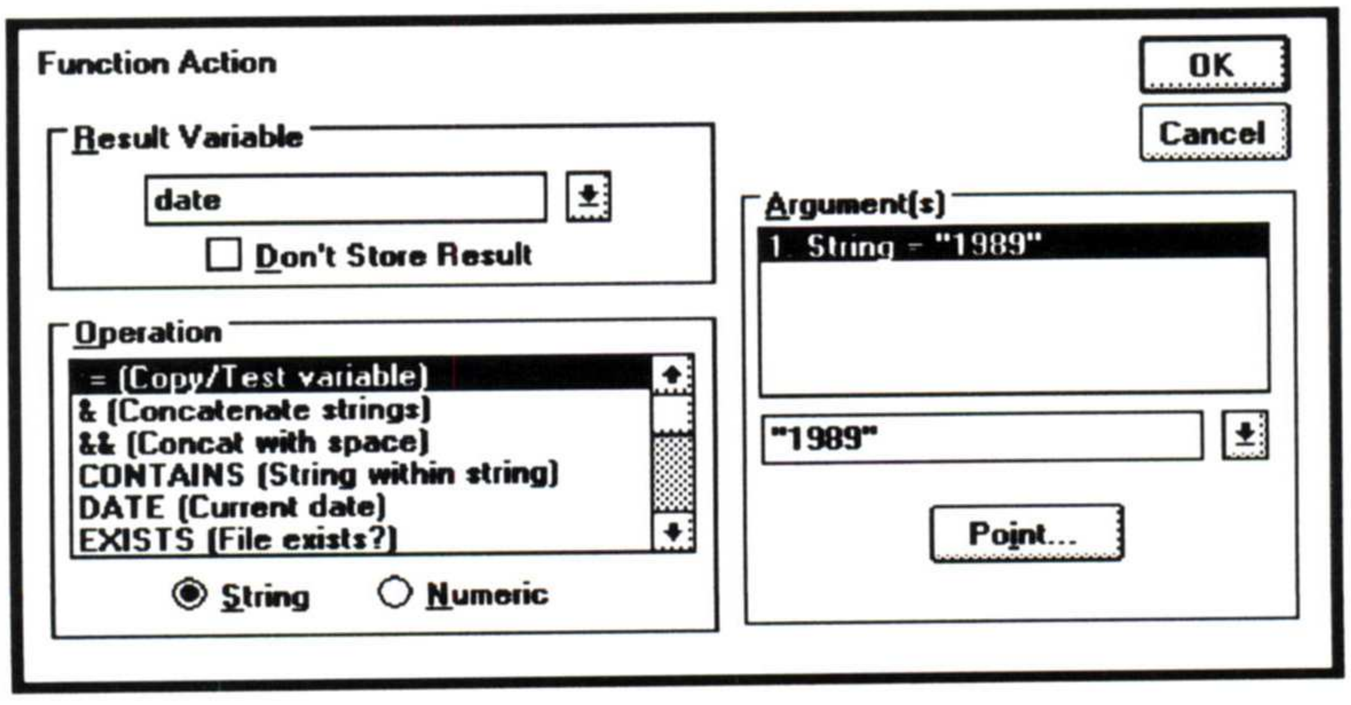

Figure 2-36 Specifying Date as a Function Action

7. Choose OK.

The Function action appears in the Actions list box.

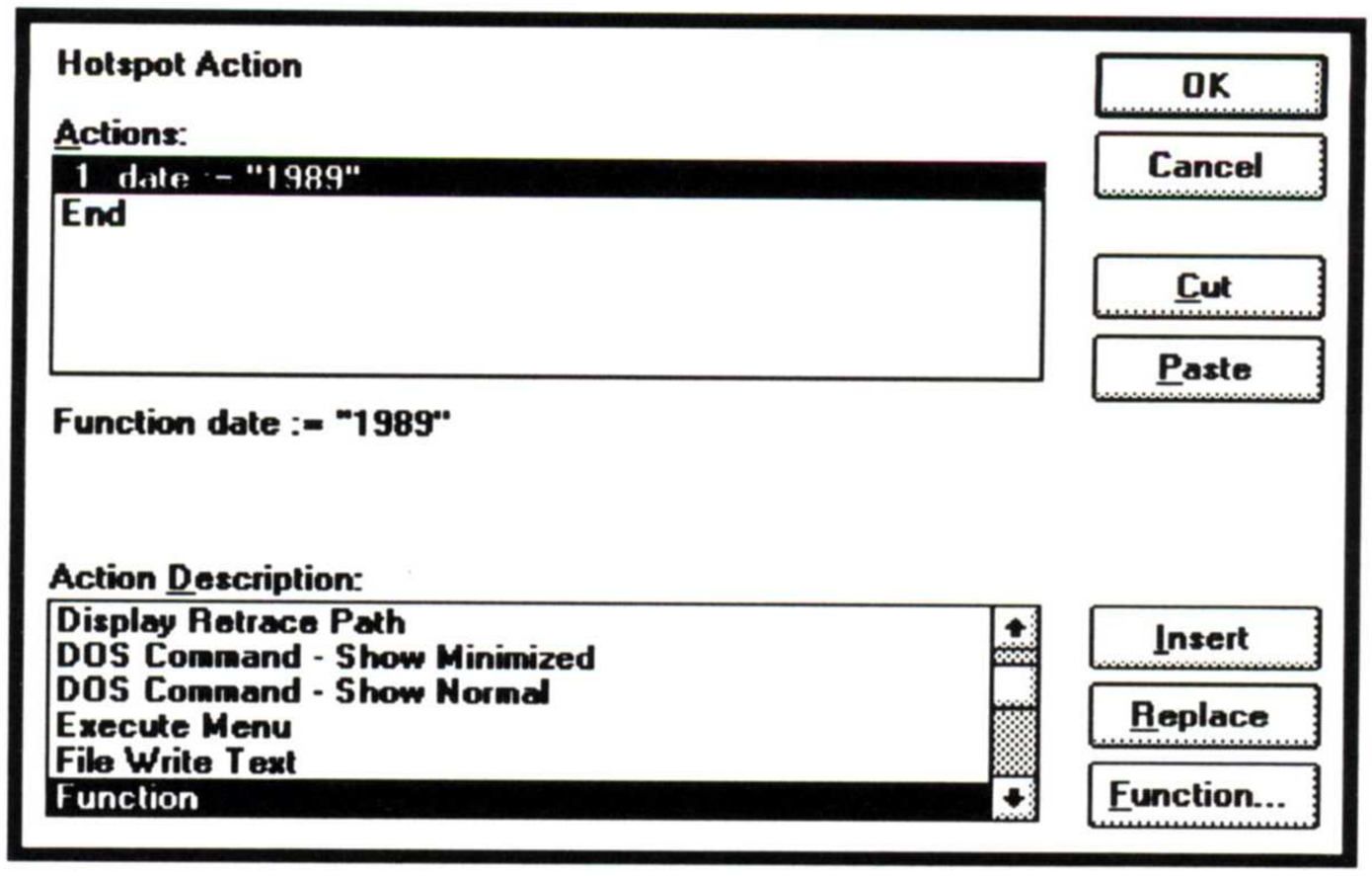

Figure 2-37 Assigning a Date

8. Choose OK.

❑ To annotate the hotspot:

1. Choose Tool Text and create a text object next to the option button.

   By default, the text is black and the background is transparent. These are the colors you want, so you can leave them alone.

2. Choose Text Font or press **F6**.

   A dialog box appears.

3. Make the text font Helvetica 12 point.

4. Choose Text Text and type:

   1989

The screen should appear as shown in Figure 2-38.

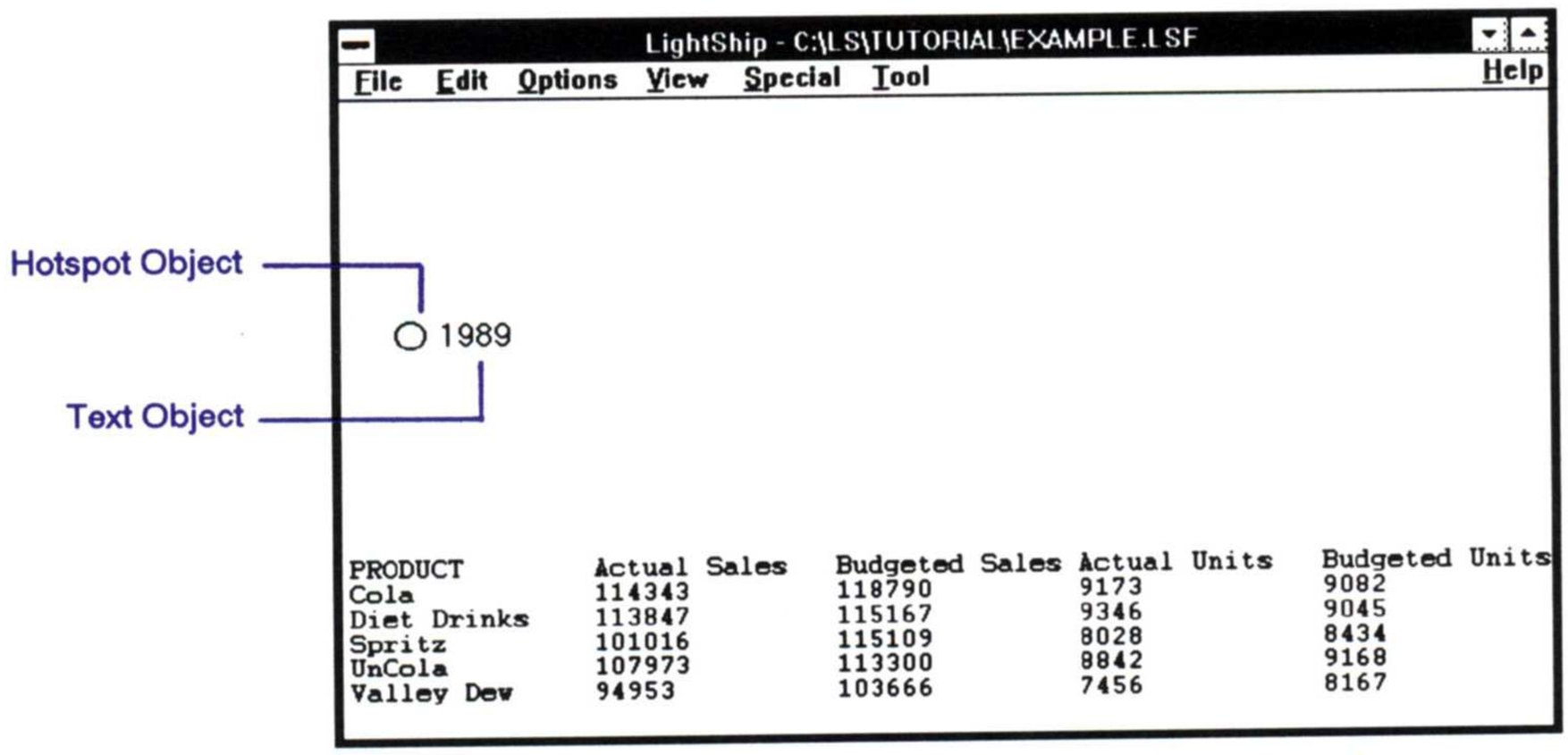

Figure 2-38 Creating the First Hotspot and Text Object

❑ To create the second hotspot:

1. Drag the mouse around the hotspot and text object to select them both.

2. Choose Edit Duplicate or press **INS**.

3. Move the duplicated objects below the first set of objects.

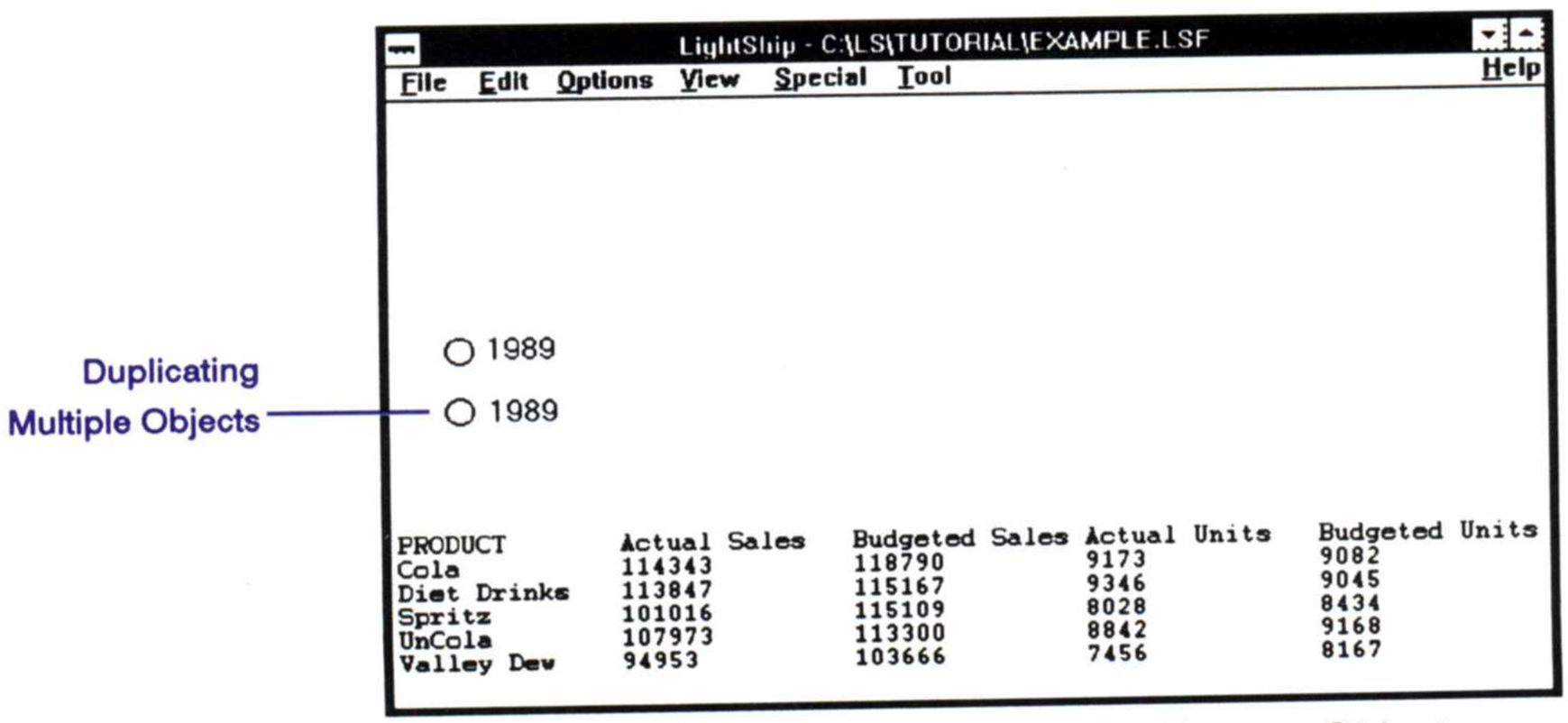

Figure 2-39 Creating the Second Hotspot Object

4. Click anywhere on the screen so that the objects are not selected.

5. Select the second hotspot.

6. Choose Hotspot Actions.

   A dialog box appears, similar to Figure 2-35.

7. Choose the Replace button.

   A dialog box appears, similar to Figure 2-36.

8. Change the Copy/Test Function action to:

   ```
   Date := "1990"
   ```

9. Choose OK twice to exit from both dialog boxes.

10. Select the second text object and change its text to:

    ```
    1990
    ```

11. Choose OK.

    Your screen should appear as shown in Figure 2-40.

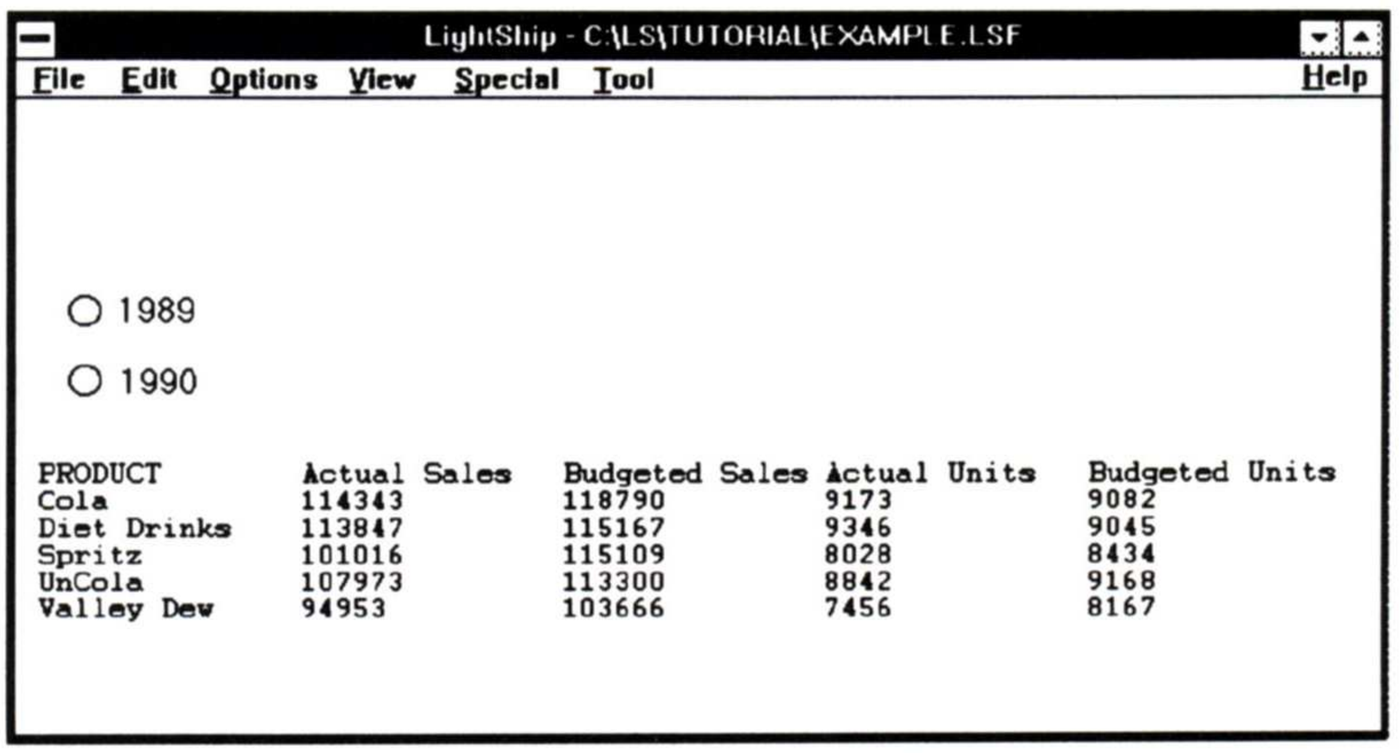

Figure 2-40 The Tutorial Screen

Browsing Through the Application

❑ **To try out the screen:**

1. Choose Special Author or press F4 to go to the Browse level.

2. Choose the 1990 button. The display is updated to reflect the date you choose.

1990 Button is Turned On to Display 1990 Data

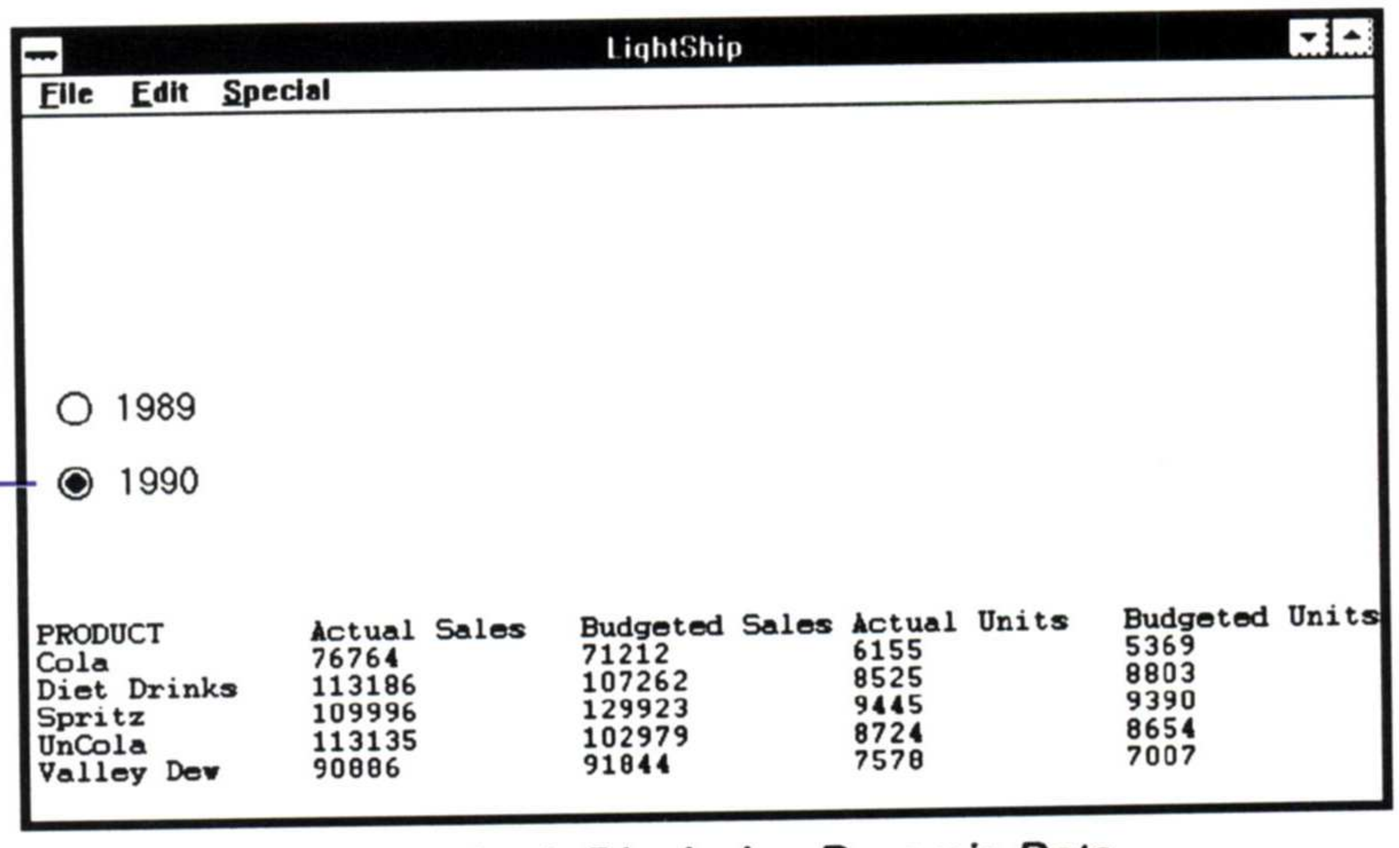

Figure 2-41 Displaying Dynamic Data

If You Have Problems

If your document object does not change when you select a hotspot, check the following:

- In LightShip, make sure the values assigned to *date* are correct for each hotspot. The hotspot labeled 1989 should assign **`date := "1989"`**. The hotspot labeled 1990 should assign **`date := "1990"`**.

- In Lens, make sure you loaded the DATE dimension with the Database Load Fields command.

- In Lens, make sure you displayed the DATE dimension correctly. Check that DATE is a displayed field in the Results command. Check that **`DATE Equal @(date)`** is a condition in the Conditions command. If the condition does not appear, it is possible that you typed the condition and chose OK without pressing Add. If this is the case, type the condition again and press Add.

- Make sure the hotspots both use the Hotspot Highlight Group 1 command. This way, only one hotspot will be highlighted at a time so you can know which year's data you are viewing.

- To see if the hotspots' *date* variables are being assigned correctly, open the Debug window by choosing Special Debug. For more information, refer to the *LightShip User's Guide*.

Chapter 3

Selecting the Database Source

This chapter explains how to select, load, and change the database source.

Selecting the Database Source

To select the type of database and the name of the file or table to load, choose the File New command. A dialog box appears.

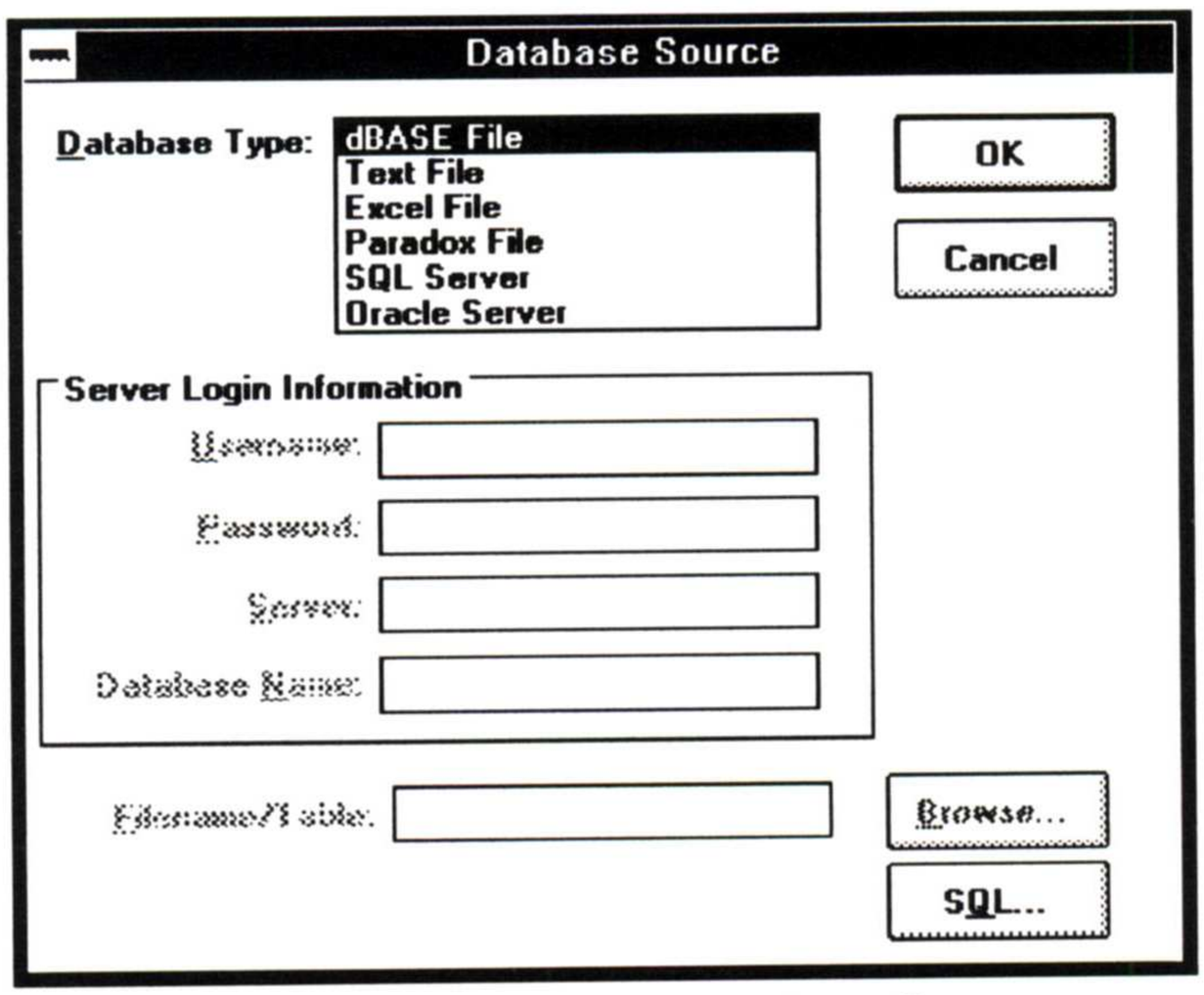

Figure 3-1 Selecting the Database Source

You can select the data to load in any of these ways using this dialog box:

- Specify a database filename or table.
- Type an SQL Select statement.

☞ **Note**

Typing an SQL Select statement is useful when especially complex criteria are necessary to describe the data to load, such as in an SQL Join operation.

Document objects save the load and display commands that you assign. If the document object is already displaying Lens data, the File New command cancels any existing load and display commands so that you can select a new source of data.

Selecting the Database Type

You can select any database type that is listed in the Database Type list box.

Logging onto a Server

The Server Login Information boxes becomes available only when you select a server in the Database Type list box, for example, SQL Server or Oracle Server.

You can type in the server login information manually or automate the login process by adding a server section to your LS.INI file. If you do not provide the necessary server login information, Lens prompts you for the information it needs.

See your system administrator for information about your username, password, server name, and database name.

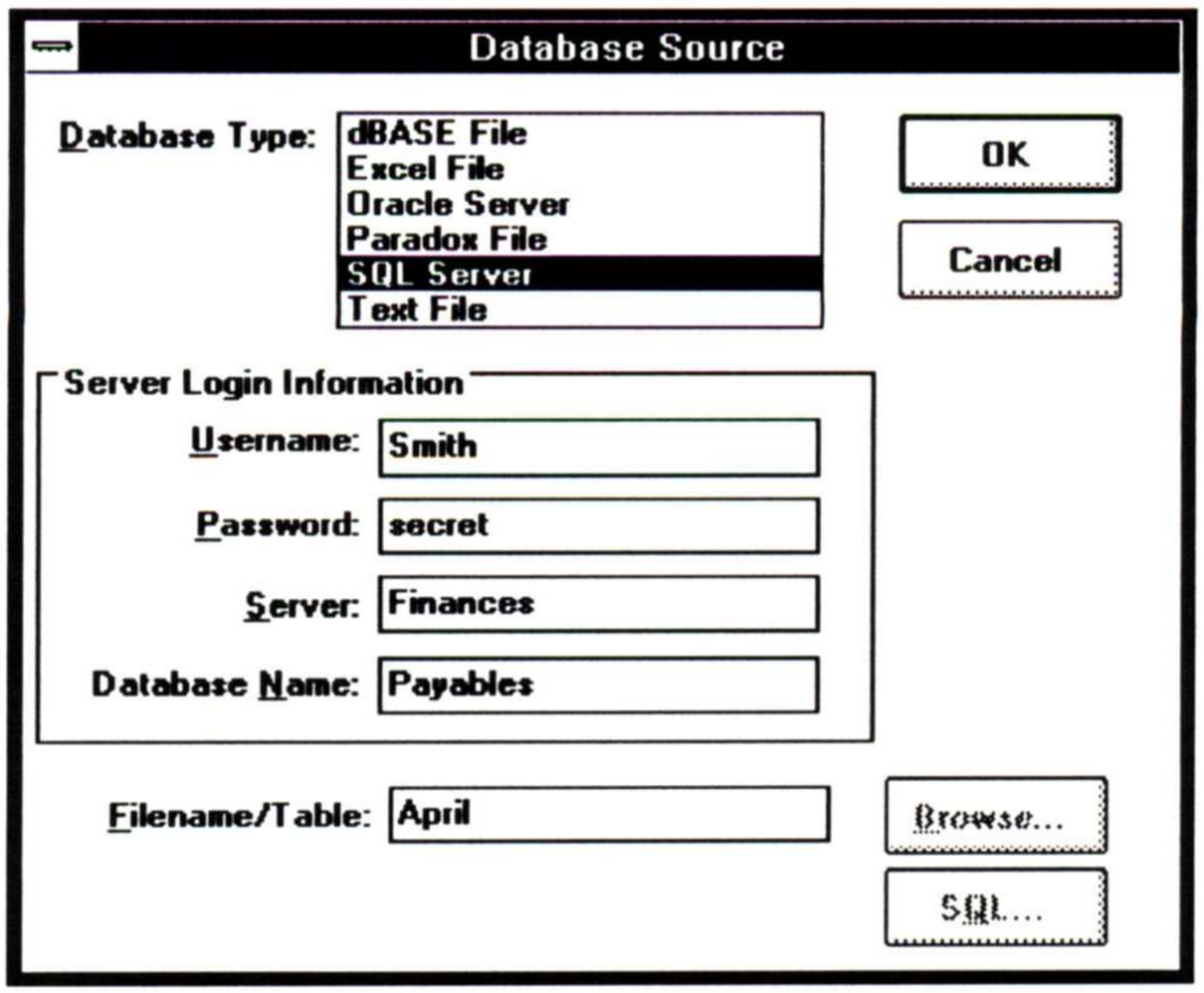

Figure 3-2 Specifying Server Login Information

Logging on Automatically

To have Lens fill in the Server Login Information boxes and log onto the server automatically, you can add the following information to your LS.INI file:

```
[<drivertype>]
Username=<username>
Password=<password>
Server=<servername>
```

where **drivertype** is the special name for the database driver you are logging on to. To determine the driver name, see the [SQL Drivers] section of your LS.INI file.

If you provide some but not all of this information, Lens will prompt you for the information it needs to continue when you select the server type from the Database Source dialog box.

For example, you might add this section to your LS.INI file so that Lens will automatically log onto the Oracle server when you select it as a database type:

```
[QLORA]
Username=Smith
Password=secret
Server=Finances
```

Selecting the Database File or Table

You specify the database file or table in the Filename/Table text box. Or you can select the SQL button to specify the file or table using an SQL Select statement.

You can choose the Browse button to search for a database file using a File Open dialog box, as shown in Figure 3-3. You can select any file based on the database type you specified.

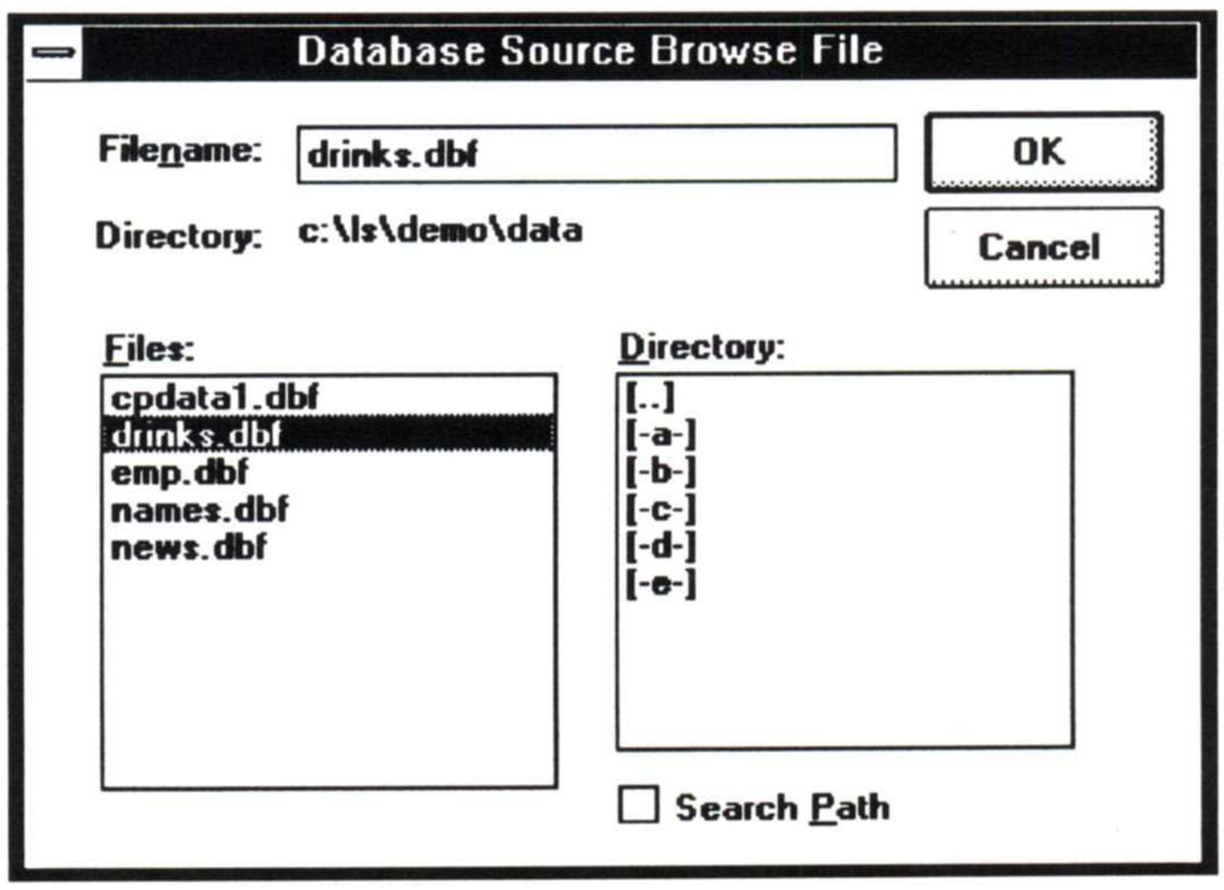

Figure 3-3 Database Source Browse File Dialog Box

For information on using the File Open dialog box, refer to the *LightShip User's Guide*.

Selecting the Data Using SQL

❑ **To type an SQL Select Statement:**

1. Select a database type from the Database Type list box.

2. Choose the SQL button.

   A dialog box appears. You can type any valid SQL Select statement for the selected database type.

If you are accessing Q+E data, you can use the Paste button to paste information from the Clipboard. For more information, refer to *Chapter 5: Using SQL Statements*.

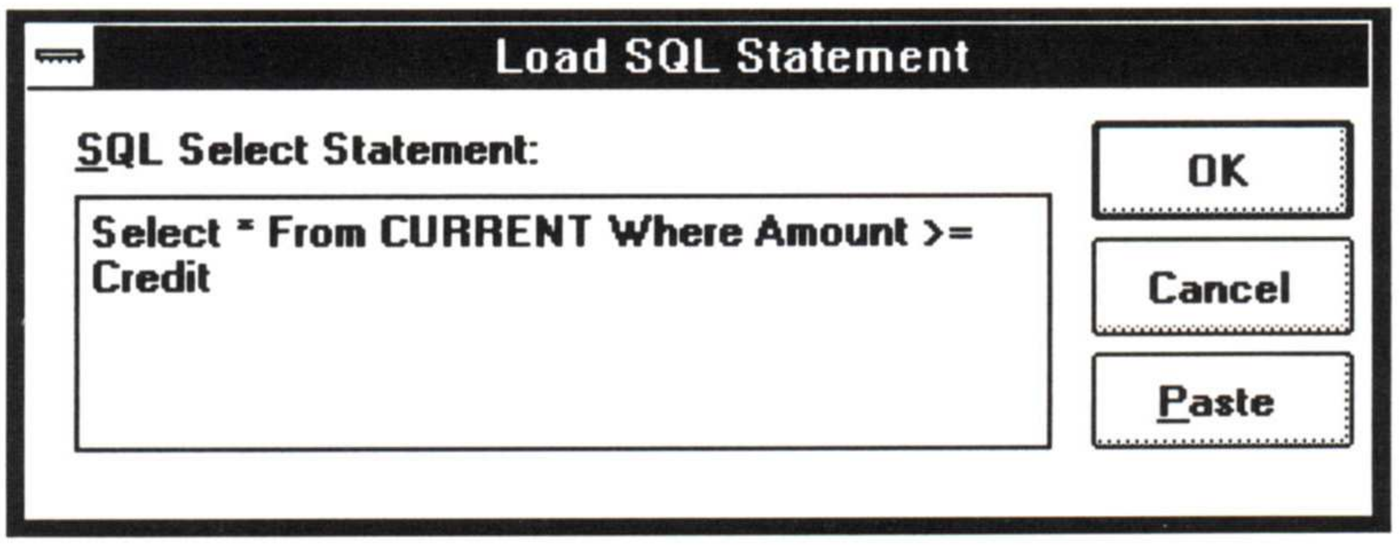

Figure 3-4 Load SQL Statement Dialog Box

Before You Load

When you specify the database source and close the Database Source dialog box, the prompt "Load the database from the source?" appears.

You can choose Yes to load the database source or choose No to defer loading until later. You might want to defer loading to further specify the load criteria in these two ways:

- Change the list of fields to load and their consolidation methods using the Database Load Fields command.

- Specify categories of data to load using the Database Load Conditions command.

Specifying Fields and Conditions

By default, Lens loads all of the fields from the database source. You can change the list of fields that Lens loads into the data cache using the Database Load Fields command.

Specifying Fields to Load

To choose the fields and their attributes for the data cache, choose the Database Load Fields command. A dialog box appears.

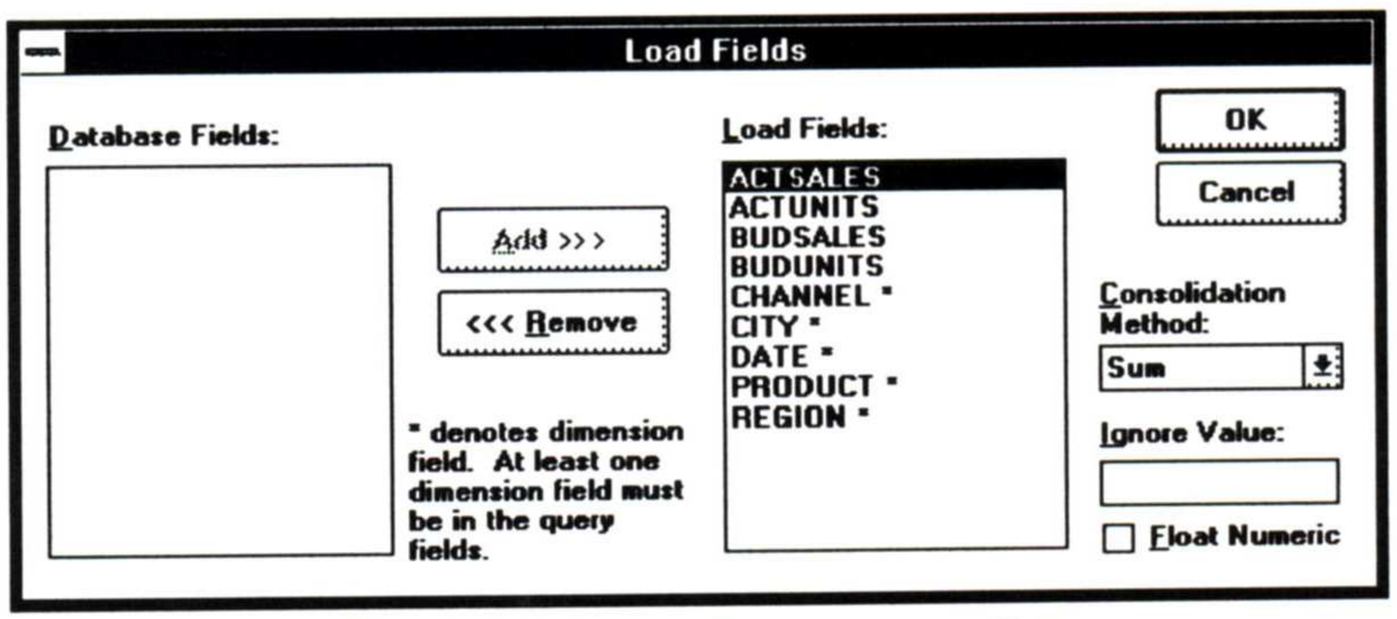

Figure 3-5 Load Fields Dialog Box

The Load Fields box lists the fields that you want to load. By default, all fields will be loaded, as shown in Figure 3-5. The Database Fields box lists the fields that you want to exclude from the data cache.

You must include at least one dimension field, denoted with an asterisk (*), in the Load Fields list.

Fields are either numeric or string. Numeric fields appear as columns of data in the Lens displays. Lens automatically consolidates numeric values according to the data and dimensions you have selected.

String fields can be up to 100 characters long. (Lens does not load character fields that are more than 100 characters long.) String fields serve as *dimensions* in Lens displays. Dimensions are indexes to the data that you can arrange horizontally or vertically.

Adding and Removing Fields

To remove a field, double-click on the field name in the Load Fields list or select the field and choose Remove.

To add a field, double-click on the field name in the Database Fields list or select the field and choose Add.

Changing a Field's Consolidation Method

By default, Lens automatically consolidates the values for each numeric field by summing its values. For example, assume that you create a display of soft drink sales figures and you:

- Load only data for the Northeast region.
- Display only data for the month of August.
- Specify Product as a dimension.

Each sales figure represents an automatic consolidation of sales for the Northeast during August for a specific product.

❑ **To change a field's consolidation method:**

1. Select the field from the Load Fields list.

   The field's current consolidation method appears in the Consolidation Method box.

2. Select a consolidation method from the drop-down list, as shown in Figure 3-6.

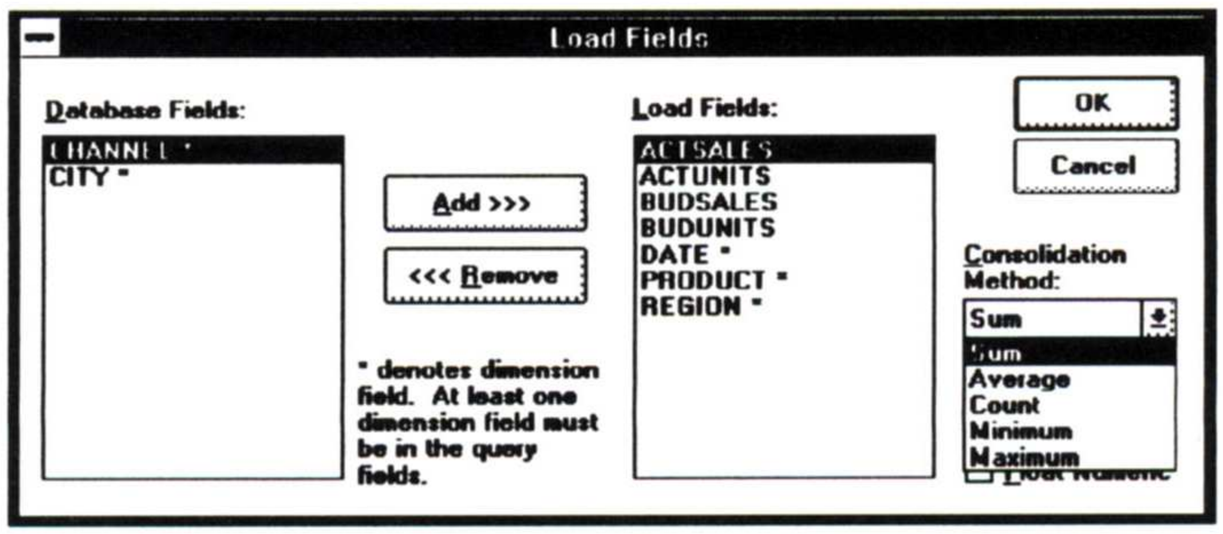

Figure 3-6 Selecting a Consolidation Method

The Consolidation methods include:

- Sum, the default, which adds the field's values.
- Average, which averages the field values.
- Count, which displays the number of consolidated records that are loaded.
- Minimum, which displays the field's smallest value.
- Maximum, which displays the field's largest value.

3. Choose OK.

Ignoring a Value During Consolidation

To avoid consolidating unavailable or missing data, you can have Lens ignore a specific value. In the Ignore Value text box shown in Figure 3-5, type one value to be ignored when consolidating the currently selected numeric field.

For example, assume a numeric value of -.0099 exists in the data to represent an N/A (not available) string for missing data points. If you type **-.0099** in the text box for the selected field, Lens will not calculate that value during a consolidation if the value is encountered.

When you select this field again, the value to be ignored appears in the Ignore Value text box.

Specifying Floating Point Arithmetic

To specify floating point arithmetic for the selected field, turn on Float Numeric. By default, Lens automatically turns on Float Numeric for the fields that are floating point or more than six digits in the database source. Lens turns off Float Numeric for integer fields in the database source.

You can do the following:

- Turn on Float Numeric to use floating point arithmetic with integer fields. You must turn on Float Numeric for integer fields whose consolidated values might exceed eight digits.
- Turn off Float Numeric to use integer arithmetic with Floating Point fields; this improves performance.

☞ Note

If you turn off Float Numeric and the consolidated result exceeds eight digits, you will receive unpredictable results without an error message.

Before You Load

When you close the Load Fields dialog box, the prompt "Load the database from the source?" appears.

You can choose Yes to load the database source into the data cache or choose No to defer loading until later. You might want to defer loading at this point so that you can further limit the data cache. If you have not already done so, you can use the Database Load Conditions command to load only specific categories of data.

Specifying Conditions to Load

To specify conditions to limit the data, choose the Database Load Conditions command. For example, you may want to load data for only a specific product and region. A dialog box appears, as shown in Figure 3-7.

☞ **Note** Although it is not normal practice, you might use the Database Load Conditions command with an SQL Select statement. Make sure that the SQL Select statement and the command do not make contradictory requests. Generally, using the Where clause in the SQL Select statement results in faster performance than using the Load Conditions command.

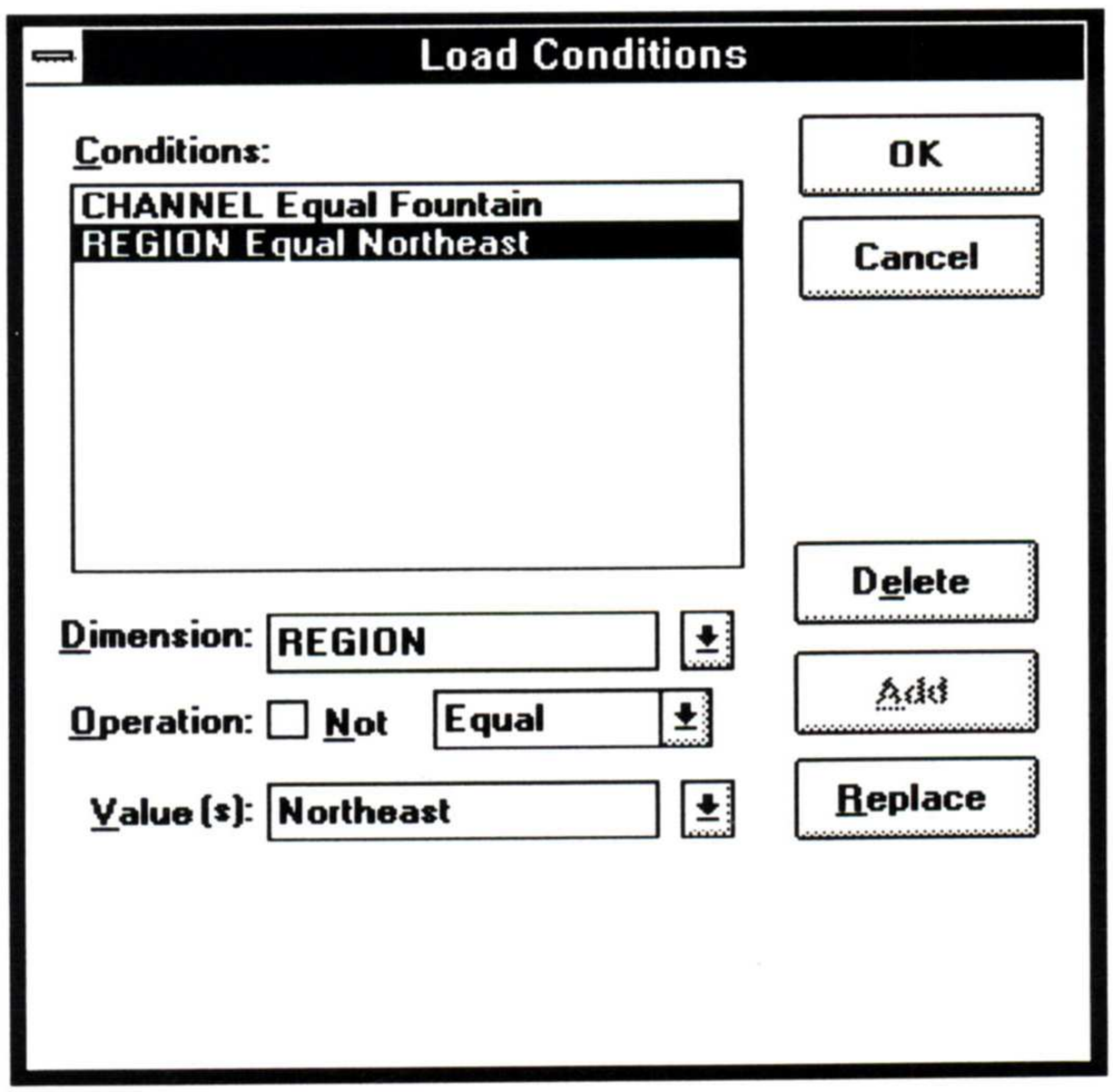

Figure 3-7 Specifying Conditions to Load

Adding and Removing Conditions

Each condition consists of an operation for a dimension field and a value. All the conditions must be satisfied for a record to be loaded.

❑ **To add a condition:**

1. Select a field from the Dimension drop-down list or type a dimension name in the text box.

2. Select an operation from the Operation drop-down list, as shown in Figure 3-8, or type the operation in the text box.

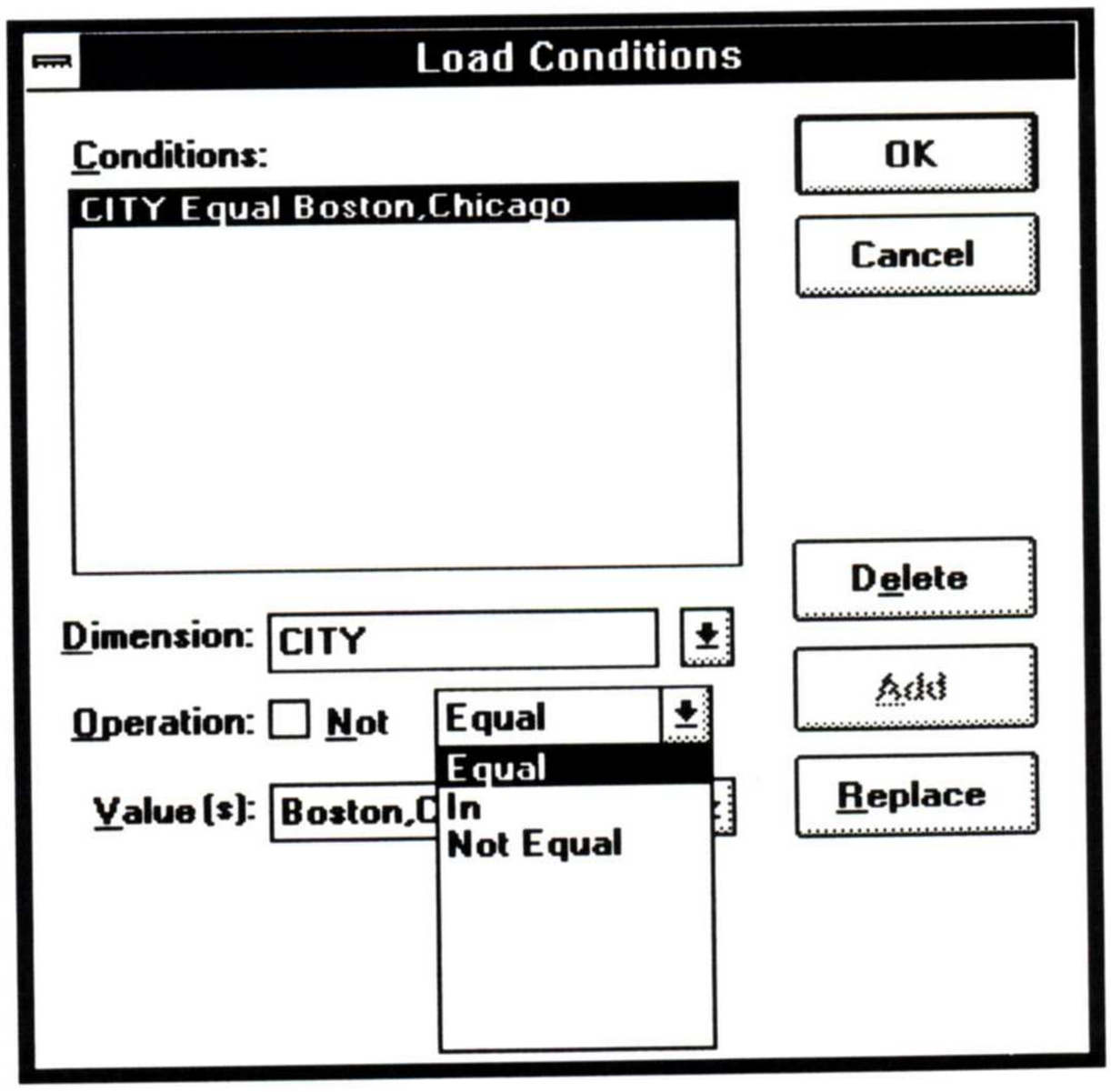

Figure 3-8 Selecting a Conditional Operation

The operations include:

- Equal, which compares values for equality such as **DATE Equal 01/05/91**.

- In, which tests that the dimension is equal to any one of the list of comma-separated values. For example, **CITY In Boston, Chicago** loads from the CITY dimension any records with values that equal Boston or Chicago.

- Not Equal, which tests that the dimension is not equal to the value.

Optionally, you can test the opposite of the operation by turning on Not.

3. Select a value from the Value(s) drop-down list or type a value in the Value(s) text box.

☞ **Note**

Character values can be any alphanumeric strings. Date values can be either: Short Date Format; Long date Format as specified in the International section of the Windows control panel; or **CCYYMMDD**, where **CC** is the century, **YY** is the year, **MM** is the month, and **DD** is the day.

4. Choose Add.

The condition is added to the Conditions list.

5. Choose OK.

❑ **To delete a condition:**

Select the condition from the Conditions list box and then choose Delete.

❑ **To edit a condition:**

1. Select the condition from the Conditions list.
2. Change the values for Dimension, Operation, or Value(s), as appropriate.
3. Choose Replace.

Before You Load

When you choose OK to close the Load Conditions dialog box, the prompt "Load the database from the source" appears.

You can choose Yes to load the database source into the data cache or choose No to defer loading until later.

You might want to defer loading at this point so that you can further limit the data. If you have not already done so, use the Database Load Fields command to change the list of fields and the consolidation methods that Lens loads from the database source.

Loading the Database Source

Once you have selected the fields and conditions to load, you can answer Yes to the "Load the database from the source?" prompt. This loads the specified data into a data cache.

Saving the Data Cache to an LSC File

Once the data is in the data cache, you might want to save it to an LSC file. To save the data cache to the currently open LSC file, choose File Save. To save the data cache to a new file or overwrite an existing LSC file, choose File Save As. A dialog box appears, as shown in Figure 3-9, so that you can select or type a new filename.

File Save and Save As are available when a data cache is loaded.

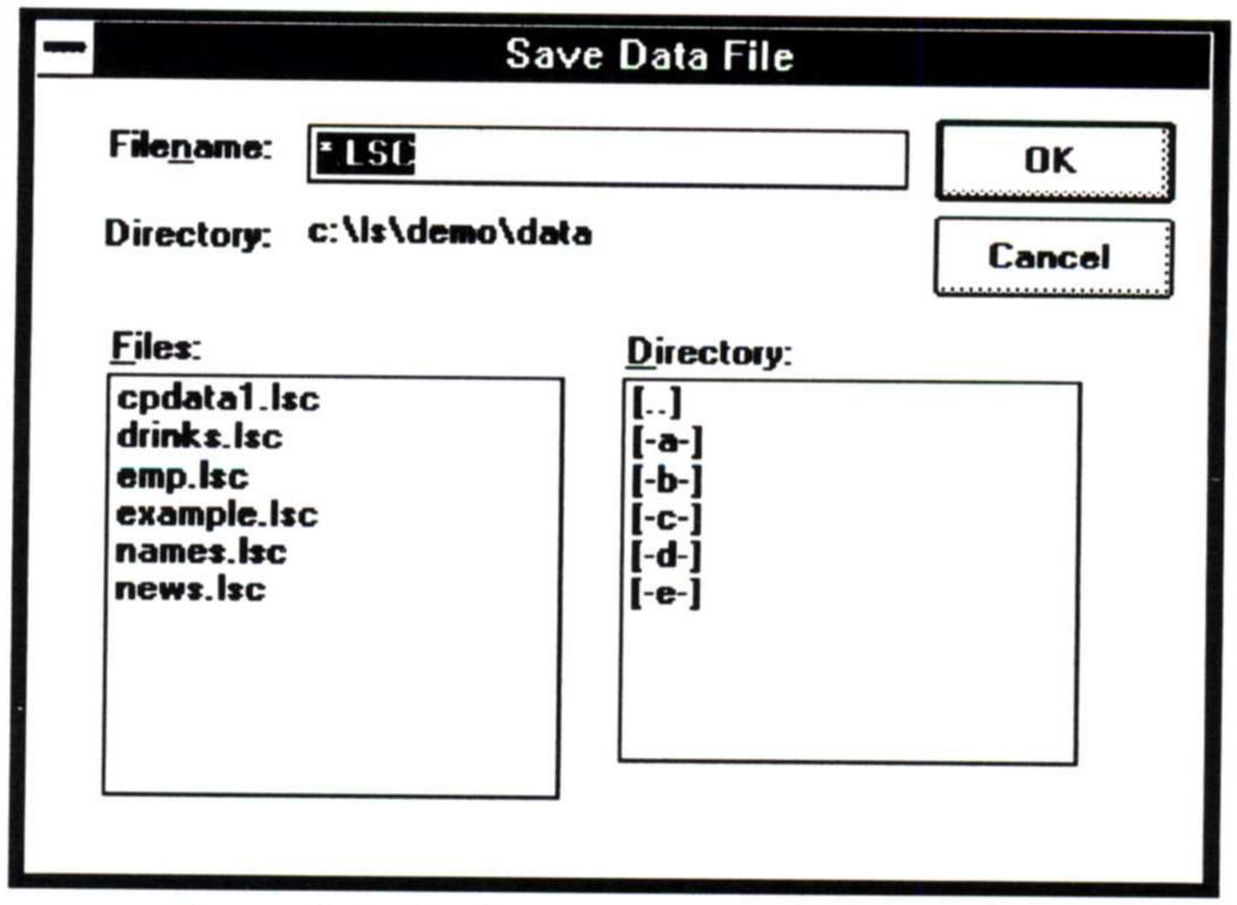

Figure 3-9 Saving the Data Cache to a File

If the document object uses an existing LSC file, its filename appears in the Filename text box. If the document object uses a direct query, ***.LSC** appears in the Filename text box.

For information on this dialog box, refer to the *LightShip User's Guide*.

Changing Sources and Displays

You can change the database source, the data cache, or the LSC file at any time.

Changing the Database Source

To change a database source, you can use either the Database Source or the File New command. The Database Source command selects a source with the same Load specifications. The File New command selects a source and cancels all existing Load specifications.

Changing a Source with the Same Load Selections

To select a different database source that has the same load selections as the current one, choose Database Source. A dialog box appears so that you can change the current source selection.

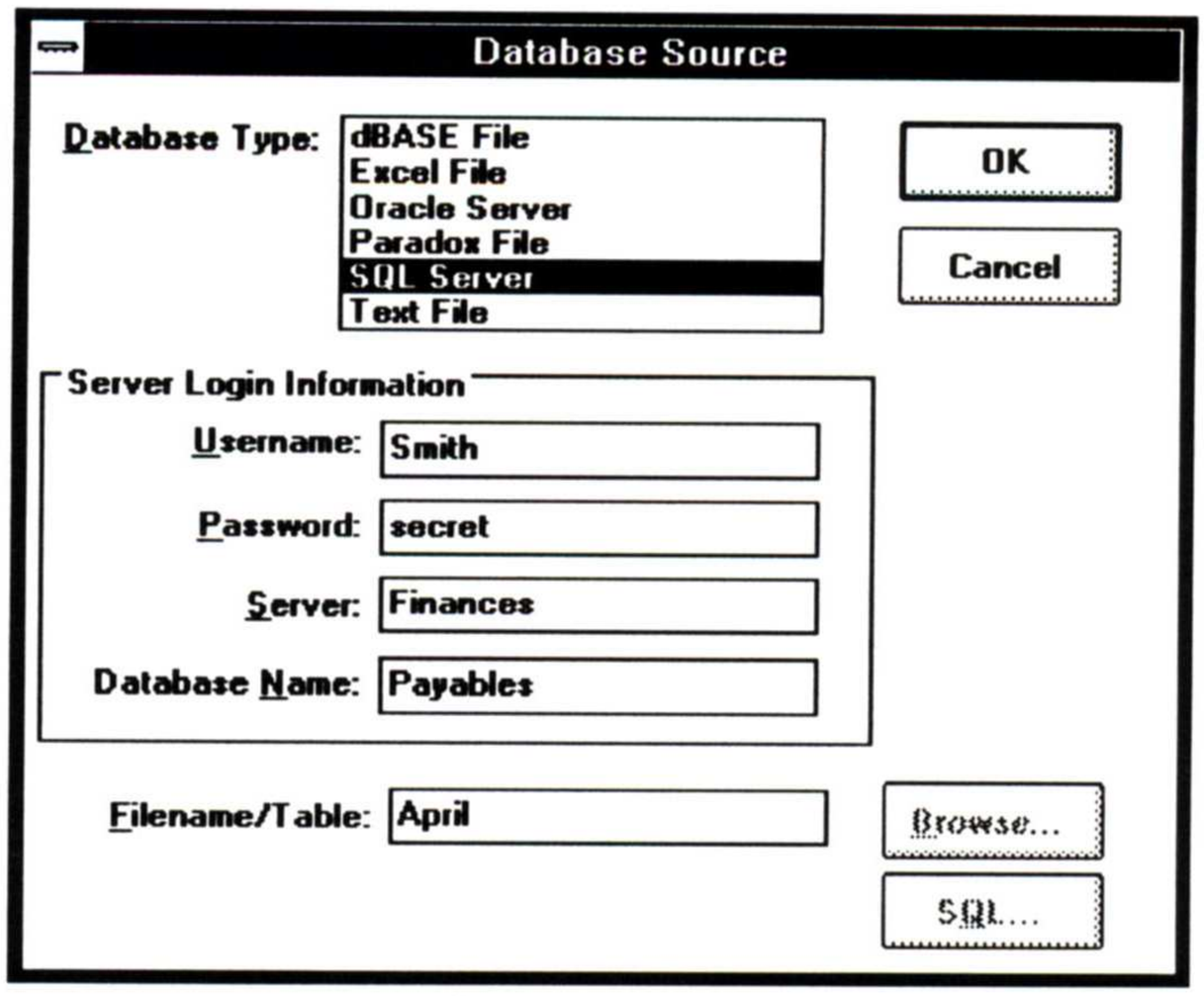

Figure 3-10 Changing the Database Source

For example, assume you have a set of dBASE files, each with the same load selections, but each containing data for a different region of the country: NESALES.DBF for Northeast data, SWSALES.DBF for Southwest data, etc. You can exchange files whenever appropriate to show different information within the same document object.

Lens maintains all the previous load specifications and attempts to load them for the new source file or table. If Lens cannot find the same fields or conditions, it displays a warning message and does not load the data.

You can adjust the display specifications using the Results and Conditions commands if the previous source displayed different fields or conditions. For more information, refer to *Chapter 4: Displaying the LightShip Lens Data*.

Changing a Source with Different Load Selections

To cancel all existing source, load, and display specifications for the document object, choose File New. It opens a dialog box similar to the one shown in Figure 3-10 except that it has no current selections.

You can load and display an entirely new database type and database source, and new fields and conditions.

Changing the Data Cache

To change the fields and conditions in the data cache, you can change the Database Load commands and then load the data to update the data cache.

The data cache must contain all the data referenced by the current display specifications. Lens warns you if the loaded data cache no longer contains the fields expected by the display. In this case, you might need to change the Results and Conditions specifications.

❑ **To change the data cache:**

1. Select the document object and choose Document Source LightShip Lens.

   The Lens window displays the current data cache.

2. Choose Database Load Fields or Database Load Conditions.

   A dialog box appears.

3. Change the load specifications as needed.

4. Choose OK.

5. At the "Load the database from the source?" prompt, choose Yes.

The new information is loaded from the database source into the data cache.

6. Optionally, you can choose File Save to save the revised data cache to an LSC file.

Changing an LSC File

You can switch from one LSC file to another as long as the load specifications remain the same. You can also update an LSC file with new data from the database source.

Use caution when changing an existing LSC file because it might invalidate the display specifications for other document objects that use this LSC file.

Switching Between LSC Files

❑ **To switch between LSC files:**

1. In Lens, choose File Open.

   The Open Data File dialog box appears.

2. Select a different LSC file.

3. Choose OK.

   The Lens window displays different information.

4. Choose File Exit/OK.

   The LightShip window appears with new document object data. If the document object uses variables references from the LSC file, make sure they are updated as needed. If the Results and Conditions specifications are different, you must update them.

Updating an LSC File

To update an LSC file to reflect changes in the database source, you can:

- Choose the Database Update Data File command.
- Execute a DOS Windows command in Windows.
- Execute a *DOS Command - Show Normal* action in LightShip.

You can use the DOS command to update one or more LSC files. You must specify any variable references that have been saved with the LSC file. The command is:

```
LSLENS [/D <variable=value>...]
 /U <filename>...
```

where **variable** is the variable name that is referenced in Lens and **value** is the variable's current value in LightShip. If you use multiple variables in an LSC file, specify each one with a /D prefix before the LSC filename.

Filename is the name of the LSC file you are updating. You can update any number of files on one command line, with each filename separated by a /U prefix. A file's variables must be issued before the filename.

DOS Command Examples

This command updates an LSC file with no variable references.

```
LSLENS /U SAMPLE.LSC
```

This command updates an LSC file that contains two variable references. *@(dbf)* is in the Filename/Table text box of the Database Source dialog box and *@(price)* is in the dialog box of the Load Conditions dialog box.

```
LSLENS /D dbf=C:\LS\NEWS.DBF /D price=100
/U SAMPLE.LSC
```

This command updates two LSC files, where each one contains a variable reference.

```
LSLENS /D price=100  /U SAMPLE.LSC
/D dbf=C:\LS\NEWS.DBF  /U REGIONS.LSC
```

This command updates one LSC file that contains a variable reference and another LSC file with no variable references.

```
LSLENS /D price=100 /U SAMPLE.LSC
/U REGIONS.LSC
```

Changing the Lens Display

To display different fields, conditions, and sort orders, choose the Conditions, Results, and Sort commands. For information, refer to *Chapter 4: Displaying the LightShip Lens Data*.

When you change a Lens display, choose File Exit/OK to save the changes or choose File Exit/Cancel to cancel them.

Chapter 4

Displaying the LightShip Lens Data

This chapter explains how to display the data from the data cache by fields and conditions and how to sort the fields.

You can specify variable references in any text box to change a display dynamically during application execution.

Displaying Fields and Column Headings

To include fields and optionally specify column headings and row labels, choose the Results command. A dialog box appears.

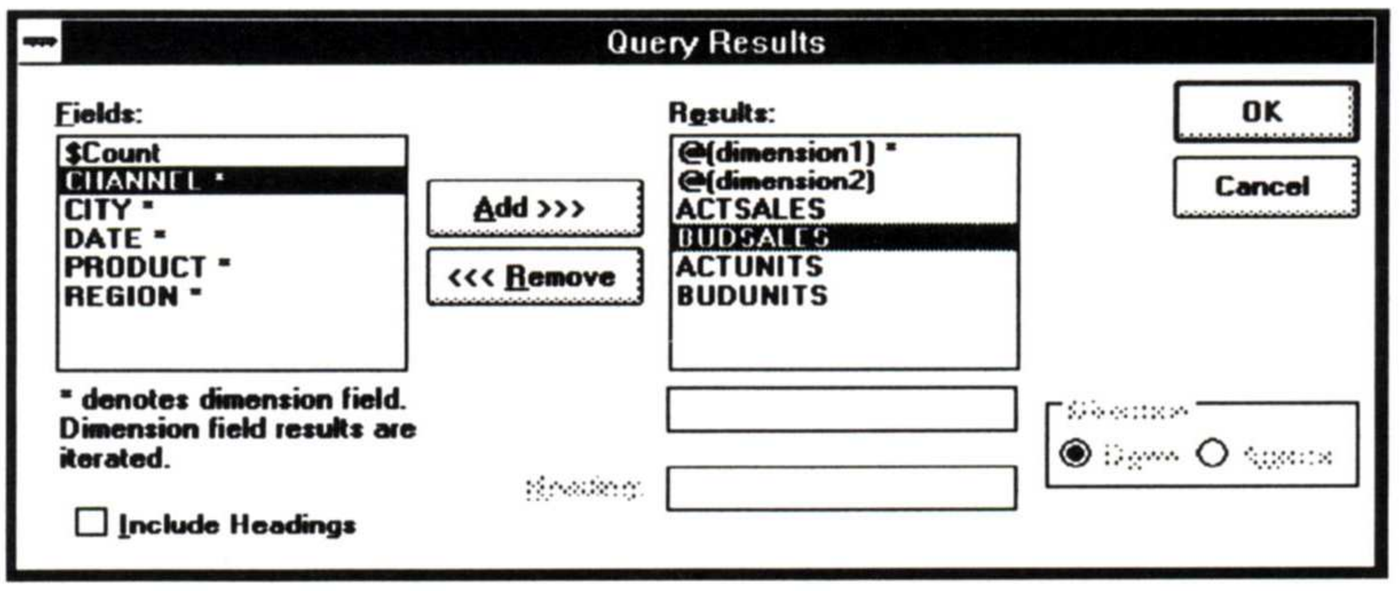

Figure 4-1 Query Results Dialog Box

Selecting Fields to Display

If the data cache is new, Lens does not display any fields. You can select the fields you want to display and you can change the display at any time.

When you select fields for display, you must consider the order that you want them to appear. The top-to-bottom order of the fields in the Results list determines the left-to-right order that fields are displayed. Since you want dimension fields to appear at either the top or left-most column of the display, you must select them first.

The numeric field at the top of the Results list will display as the first column, the next numeric field will display as the second column, and so on. You can include a $Count field that displays the number of low-level records in each consolidated record.

Displaying Fields

When you add a field, it is placed under the field that is currently selected in the Results list.

❑ **To add a field to the display:**

1. Select a field in the Results list at the point where you want to place a field.

   If this is the first time you are displaying data, remember to display a dimension field first.

2. Double-click on the fields in the Fields list, or select fields and choose Add each time.

   The Results list is updated.

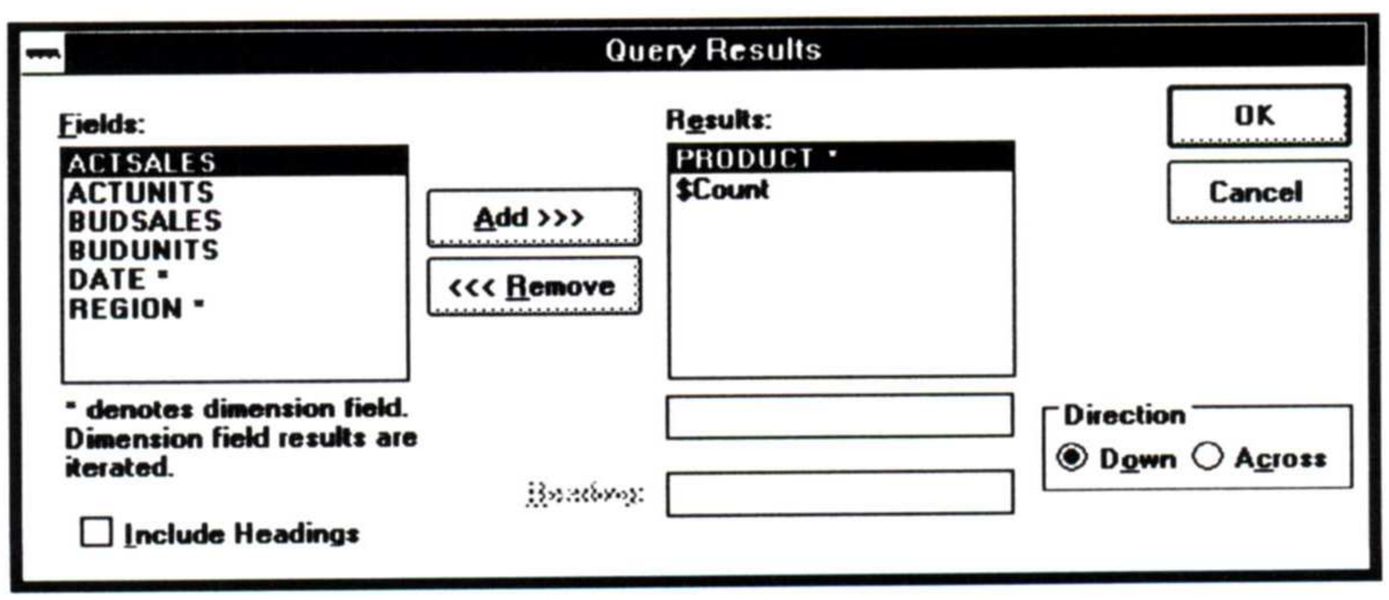

Figure 4-2 Adding Fields to the Display

3. Choose OK.

   The Lens window is updated with the selected fields.

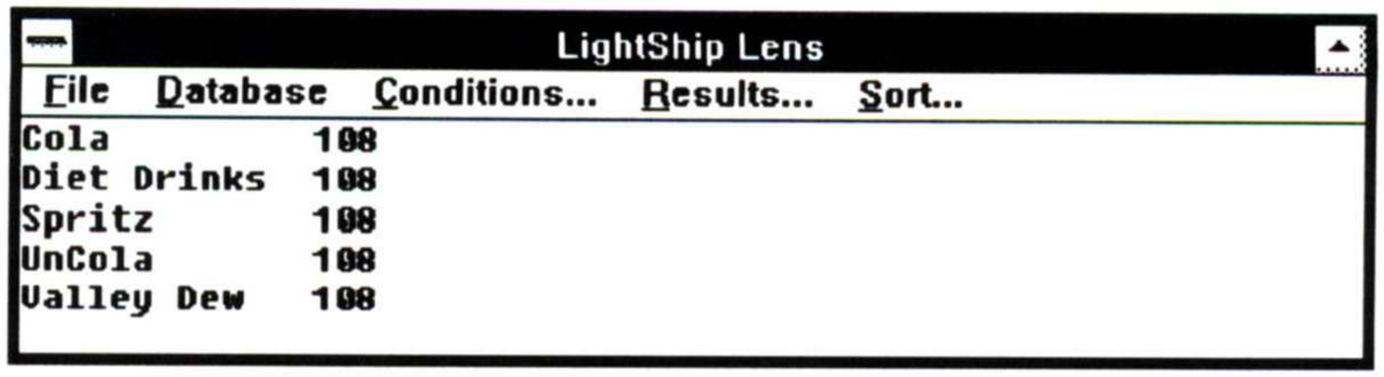

Figure 4-3 Displaying Fields In Lens

Excluding Fields

❑ **To remove a field from the display:**

1. Double-click on the field in the Results list that you want to remove, or select the field and choose Remove.

   The field appears in the Fields list.

2. Choose OK.

   The Lens window is updated based on the field that was removed.

❑ **To change the order of fields:**

Remove fields from the Results list, then add them in the positions you want.

Referencing Variables as Results Fields

To use variable references as field names, type the variable reference in the edit box beneath the Results list. You can reference a variable in any text box in Lens that allows four characters or more.

Figure 4-4 shows the Results list box and text box with a variable reference used as a field name.

Figure 4-4 Results List and Text Box

Changing the Direction of Dimensions

To change the direction of a dimension's values, select the dimension and turn on the appropriate Direction option. Across displays values in a row and Down displays values down a column.

For example, you can display the PRODUCT dimension down the left column of the screen, as shown in Figure 4-5.

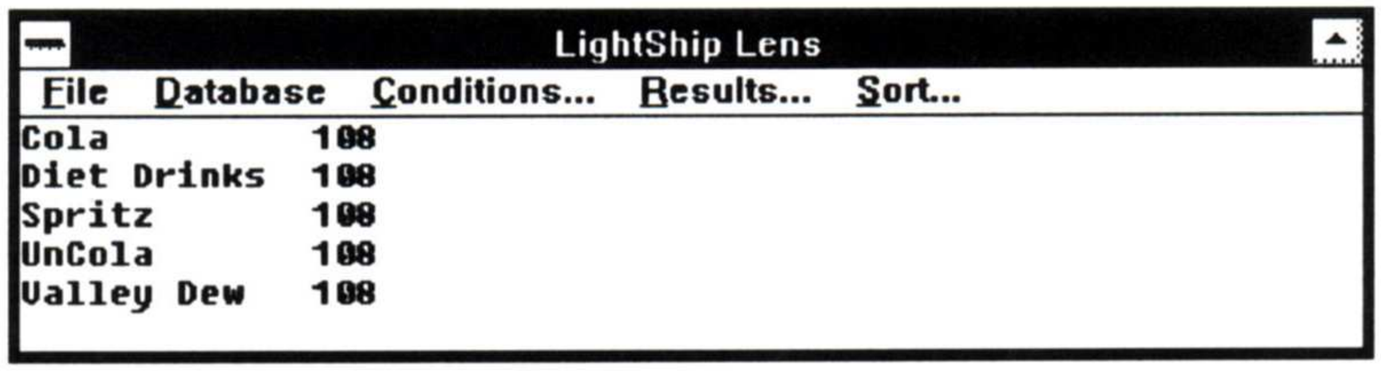

Figure 4-5 Displaying a Dimension Down a Column

You can also display the PRODUCT dimension across the top of the screen.

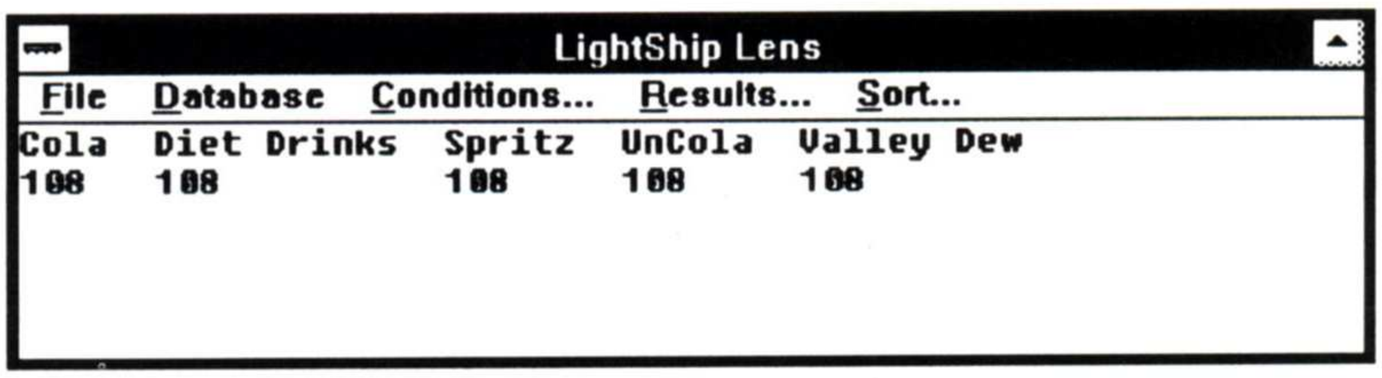

Figure 4-6 Displaying a Dimension Across a Row

Adding Column Headings

To add column headings or row labels, turn on the Include Headings option. The fields you display will also display a field name and consolidation method as the column heading.

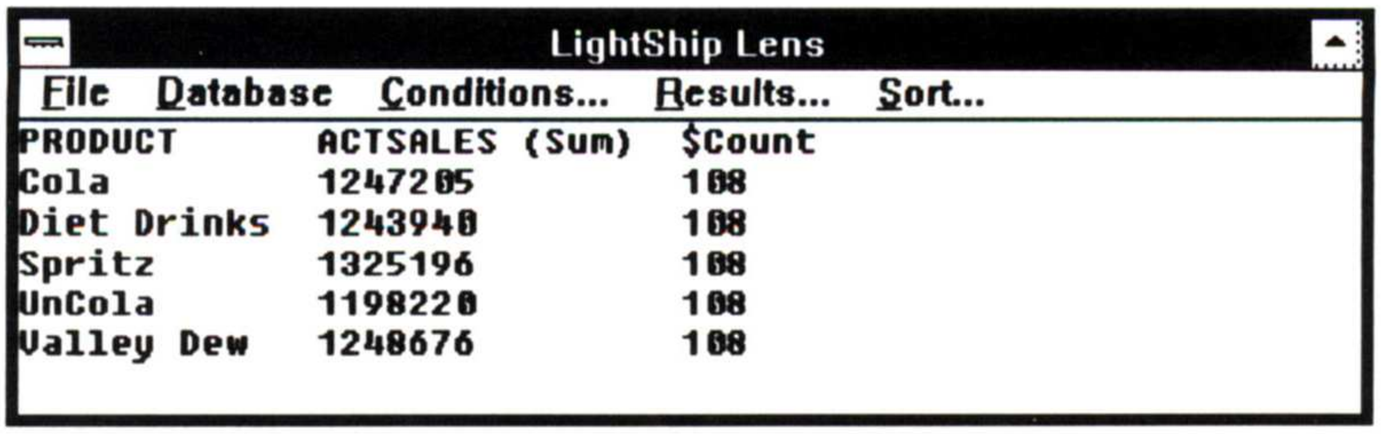

Figure 4-7 Displaying Default Column Headings

You can change a field name to a text string that is easier to read.

❑ **To change a field name to a text string:**

1. Make sure that Include Headings is turned on. Select the field from the Results list.

   If you have already specified a heading other than the field name, the heading appears in the Heading text box. Otherwise, no text appears in the text box.

2. Select the Heading text box and type an appropriate heading.

   - You can skip a column heading by typing a space.

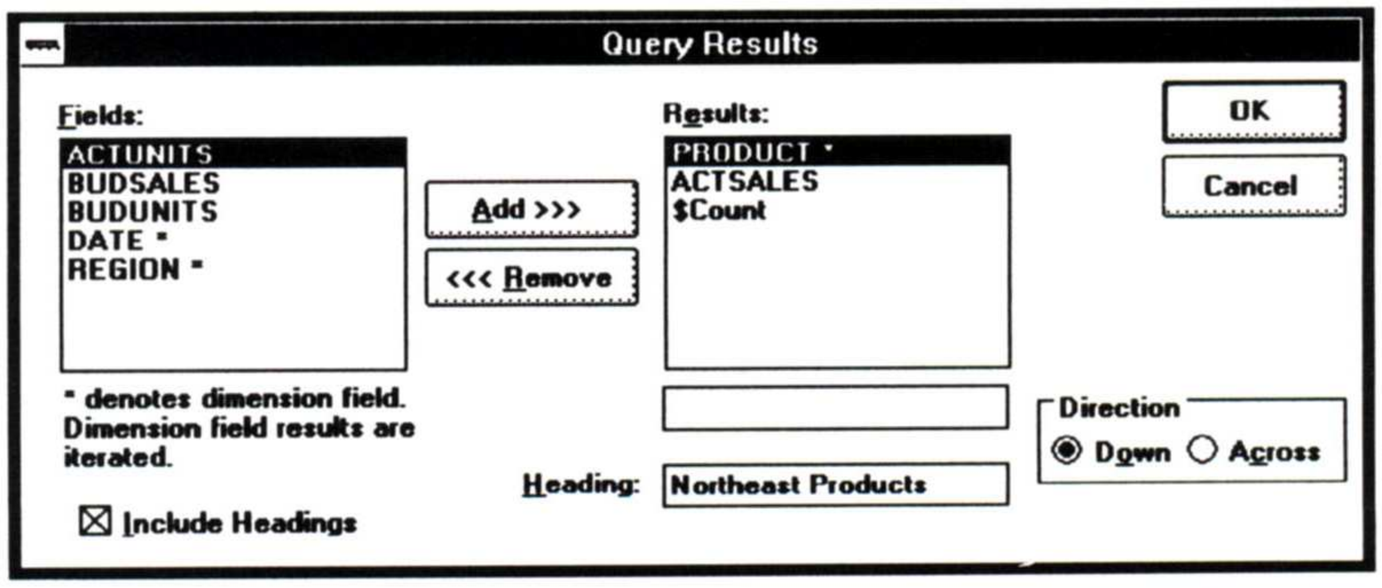

Figure 4-8 Changing the Column Heading to a Text String

3. Choose OK.

   The text appears as a heading for that field's values.

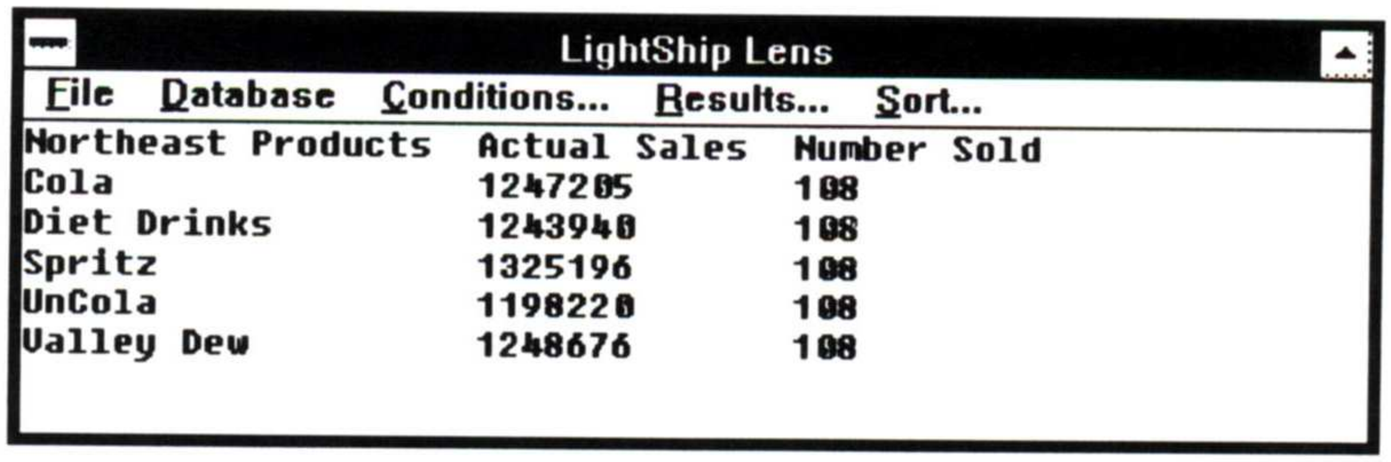

Figure 4-9 The Lens Display with Column Headings

Specifying Display Conditions

To specify criteria that limit the consolidation and display of data, choose the Conditions command. A dialog box appears.

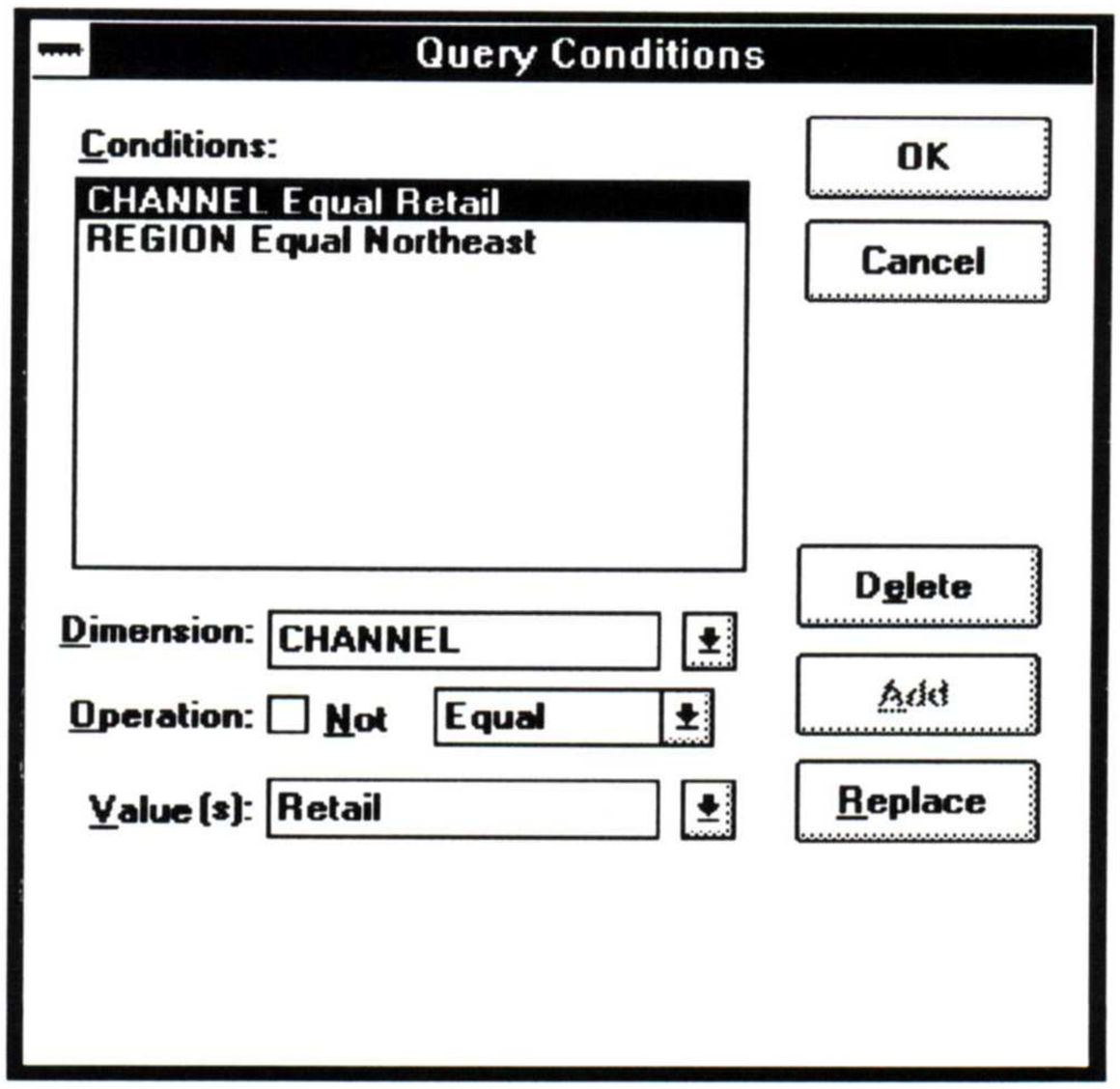

Figure 4-10 Query Conditions Dialog Box

Adding a Display Condition

You can add conditions that limit the display of data at any time. For example, assume the data cache contains records with the DATE field values ranging from **19890501** to **19910501**. If you specify the condition *DATE Like @(date)* and the variable value of *date* is **1989**, Lens displays the records with DATE fields like 1989.

❑ To add a condition:

1. Select a dimension from the Dimension drop-down list or type a dimension name in the Dimension text box.
   The list includes all the dimensions that you loaded from the database source.

2. Select an operation from the Operation drop-down list.

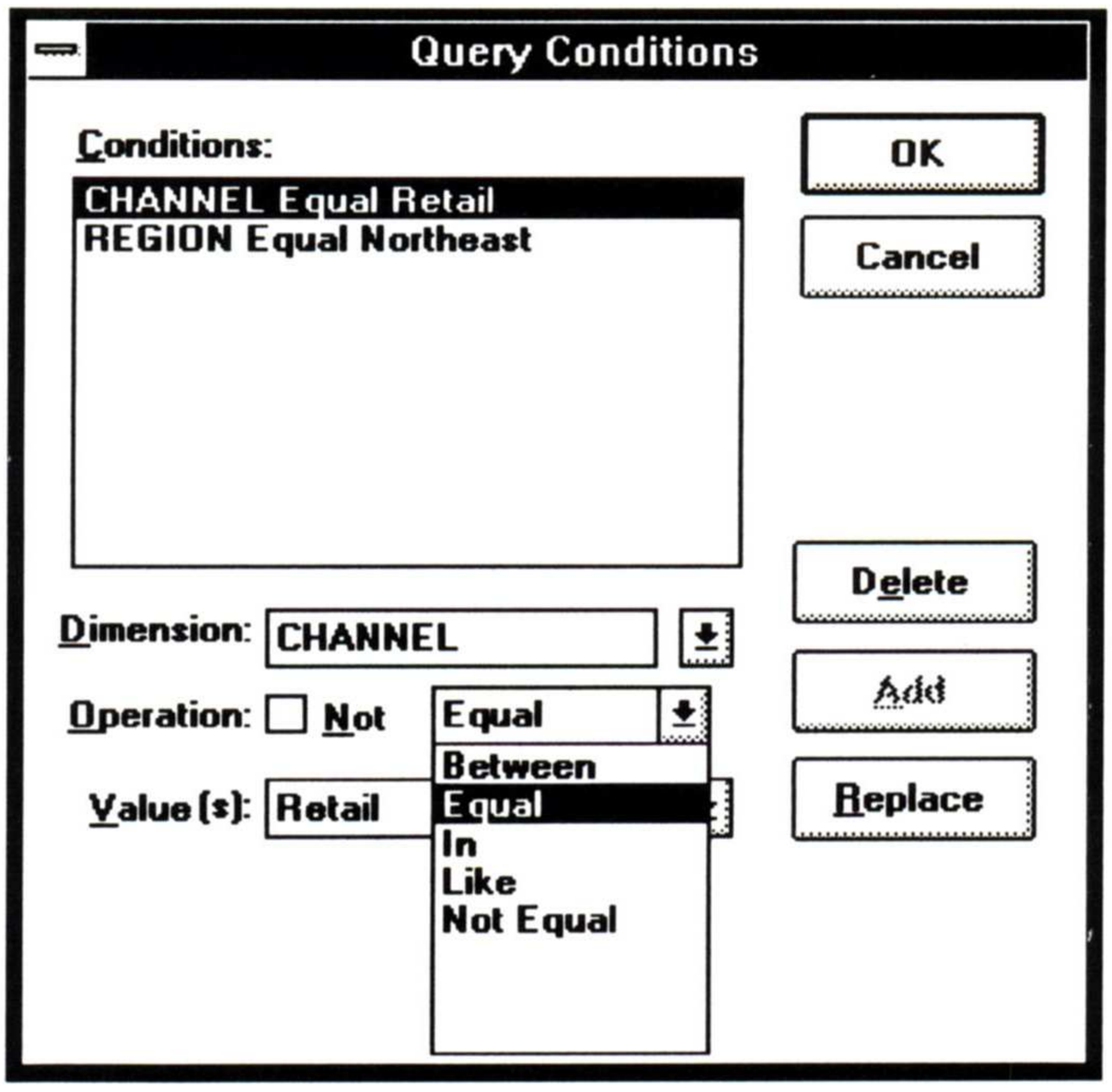

Figure 4-11 Selecting a Condition Operation

The selection of operations include:

- Between, which displays records that fall alphabetically between two comma-separated values. For example, **CITY Between B, D** displays records for the cities of Boston and Chicago. Between is inclusive, so the example would display records such as the letter "D", but would not include Dayton.

- Equal, which displays records that equal the specified value(s). For example, **REGION Equal South** displays all the South records from REGION.

- In, which displays records that exist for any specified value. For example, **CITY In Boston, San Francisco** displays any Boston or San Francisco records from the CITY dimension. You can include up to 100 values in an *In* condition, each one separated by a comma.

- Like, which displays records whose left-most portion matches the specified value. For example, **EMPNAME Like B** selects all the last names that begin with B from EMPNAME.

- Not Equal, which displays records that are not equal to the value.

- To test for the opposite of the operation, turn on Not.

3. Select a value from the Value(s) drop-down list or type a value in the text box.

☞ **Note**

Lens stores and displays date values in either: Short Date Format; Long Date Format, as specified in the International section of the Control Panel; or **CCYYMMDD**, where **CC** is the century, **YY** is the year, **MM** is the month, and **DD** is the day.

4. Choose Add.

   The Conditions box lists the condition for displaying data.

Changing a Display Condition

❑ **To remove a condition:**

Select the condition in the Conditions list box and choose Delete.

❑ **To change a condition:**

Select the condition in the Conditions list box, change the values in the Dimension, Operation, or Value(s) boxes and choose Replace.

Sorting the Lens Display

To sort the fields in a Lens display, choose the Sort command. A dialog box appears.

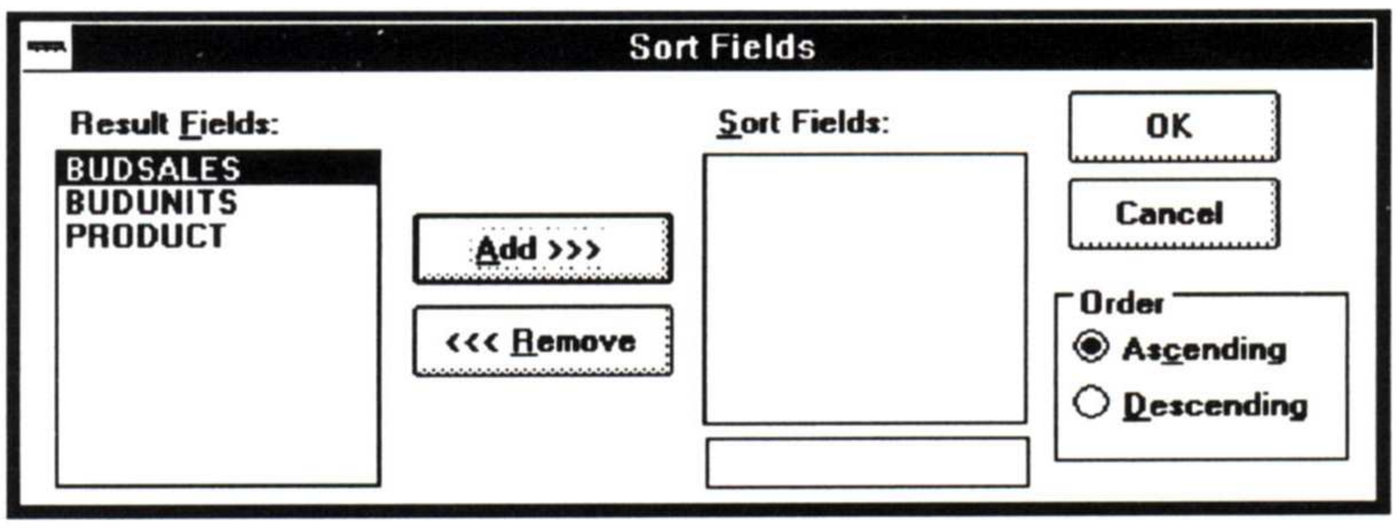

Figure 4-12 Sorting Fields

The Sort Fields list contains the list of fields to be sorted. The top-to-bottom order of sort fields determines the sort hierarchy.

For example, assume that you move BUDSALES and then BUDUNITS into the Sort Fields list in Figure 4-12. Records will be displayed in order of Budgeted Sales values, and then for any identical Budgeted Sales values, in order of Budgeted Units values. The order can be ascending or descending.

The Result Fields box lists the fields that will not influence the sort hierarchy.

The Sort command is most useful in one-dimensional displays. In multi-dimensional displays, Lens first fits data within the dimensions and then attempts to sort values within that context. If you do not specify a sort order, Lens sorts the dimension values alphabetically.

Sorting Fields

❑ To sort a field:

1. Double-click on the field in the Result Fields list that you want to sort, or select it and choose Add.
   Its current sort order is turned on in the Order box.

   The field name moves to the Sort Fields list. If there are already fields in the Sort Fields list, Lens places the field *under* the currently highlighted one.

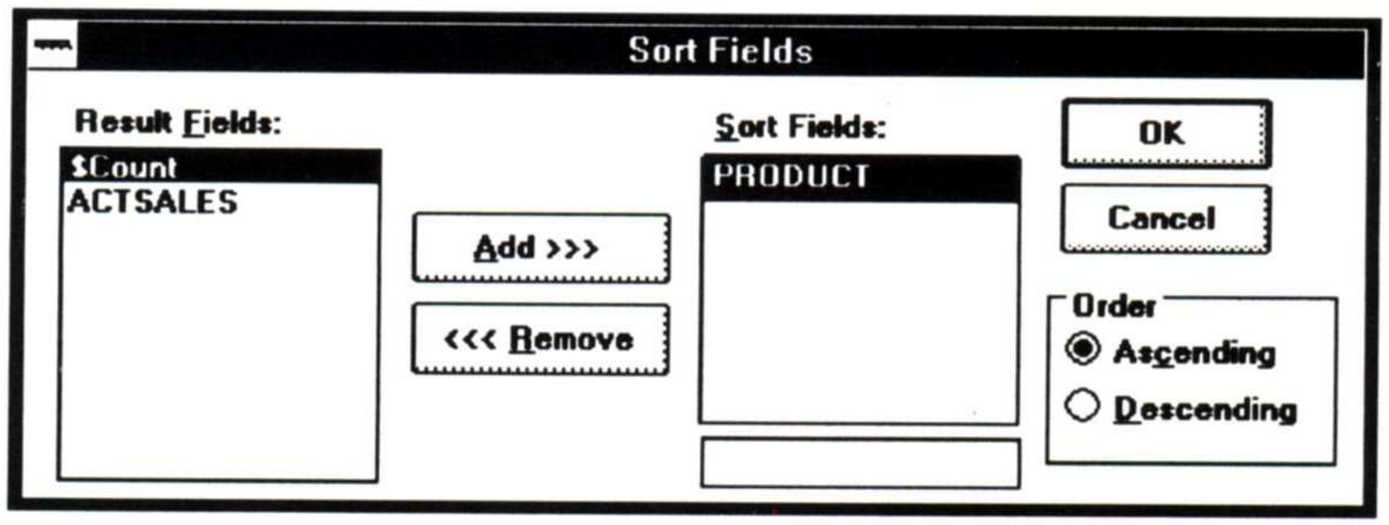

Figure 4-13 Selecting the Field to Sort

2. Turn on Descending or Ascending depending on the direction that you want to display sorted values.

3. Choose OK.

   The Lens window appears with the selected sort order. Figure 4-14 shows how the PRODUCT dimension is sorted in descending order.

Figure 4-14 Sorting Products by Descending Order

Figure 4-15 shows how the PRODUCT dimension is sorted in ascending order.

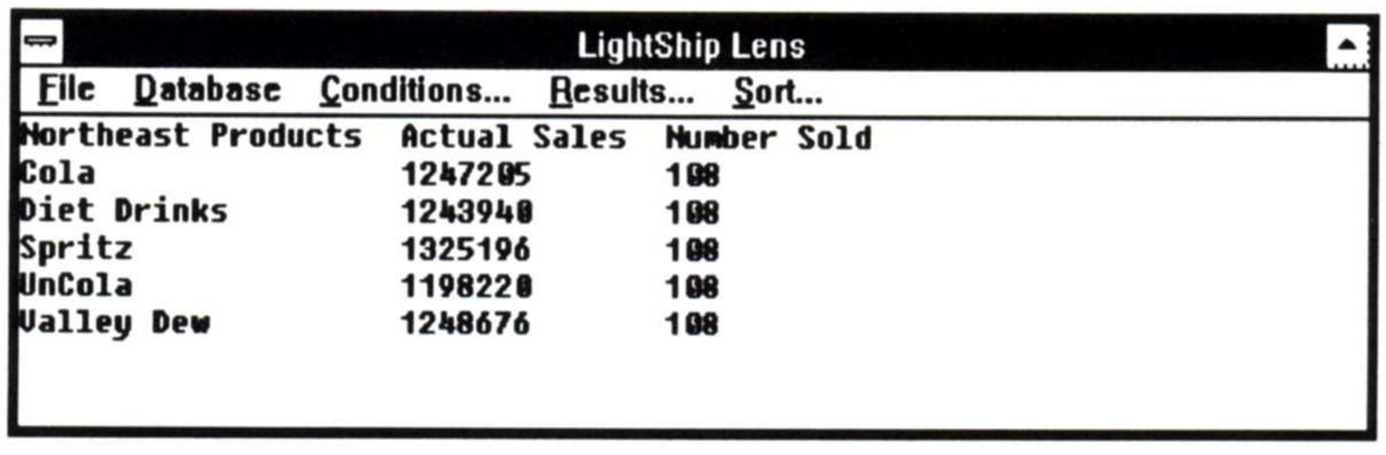

Figure 4-15 Sorting Products By Ascending Order

❑ **To remove a field from the sort order:**

Double-click on the field in the Sort Fields list, or select it and choose Remove.

☞ **Note** Lens sorts date values using the format CCYYMMDD.

Saving Changes to the Lens Display

To save the changes to the Lens display, choose File Exit/OK. To discard the changes, choose File Exit/Cancel or choose Close from the Windows Control menu.

Chapter 5

Using SQL Select Statements

The first part of this chapter describes how to enter SQL Select statements. The second part of this chapter provides information on the SQL Select statements for the database systems that Pilot supports. It is based on Pioneer Software System's *Q+E Database Library User Guide* and includes information on:

- dBASE-compatible database files
- SQL Server databases
- Oracle databases
- Text files
- Excel worksheet files
- IBM DB2 databases
- NetWare SQL databases
- Paradox databases

For more information about these external programs, refer to the User Guide of that product, or refer to that company's Customer Support department.

Specifying an SQL Select Statement

Rather than select a database table or file as the database source, you can type an SQL Select statement.

Typing an SQL Select Statement

❑ To type an SQL Select statement in Lens:

1. ■ Select a new database source by choosing File New or,
   ■ Switch database sources by choosing Database Source.

   A dialog box appears.

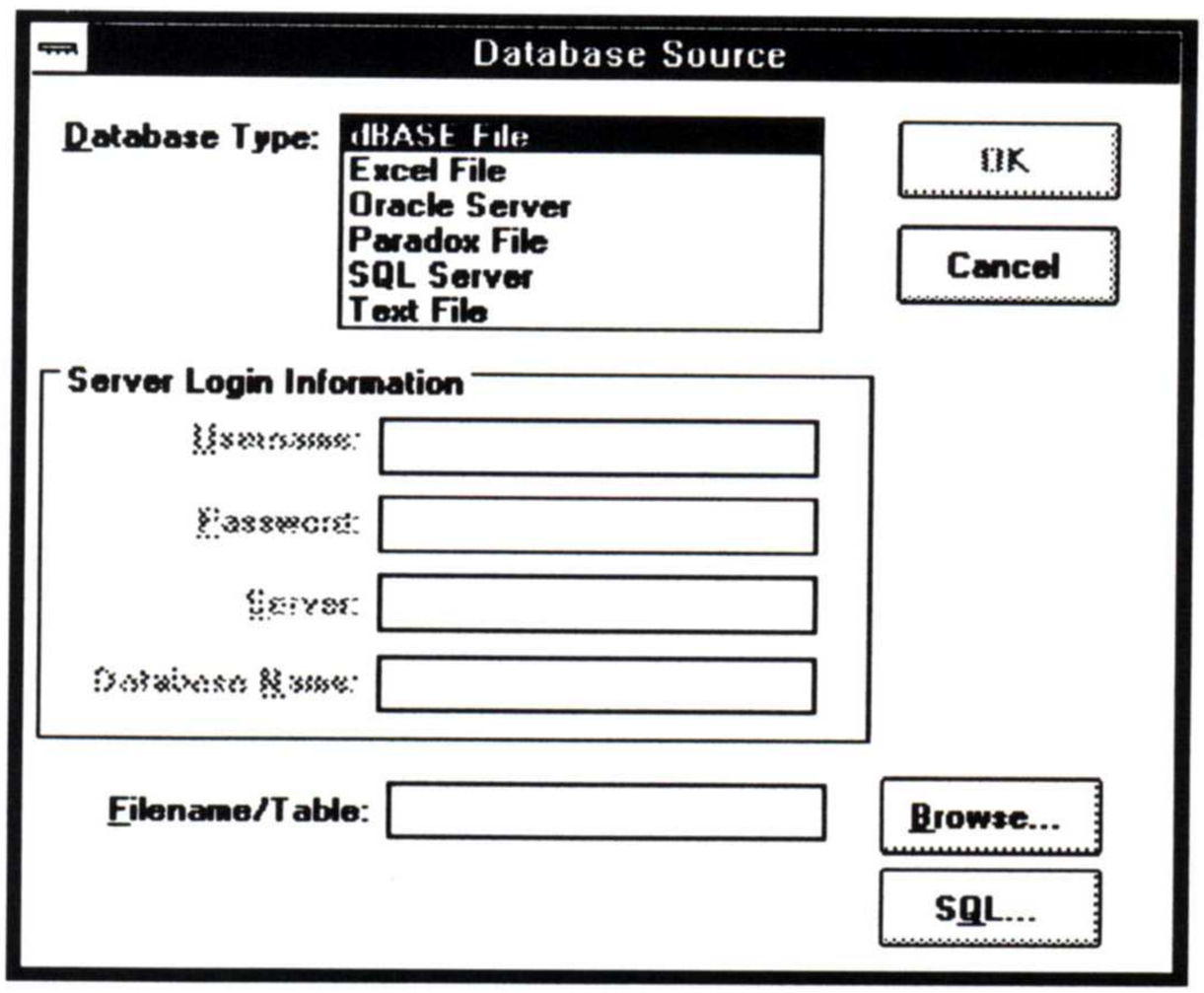

Figure 5-1 Selecting the Database Source

2. Select the type of database from the Database Type box.
3. Choose SQL.

   A dialog box appears, as shown in Figure 5-2.

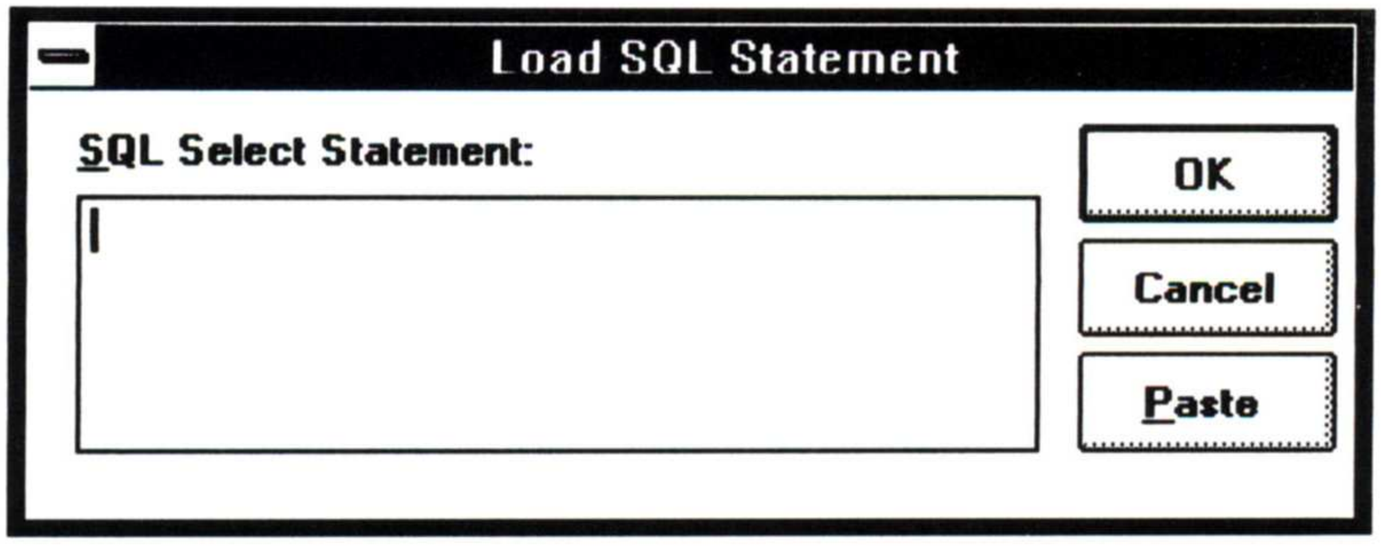

Figure 5-2 Load SQL Statement Dialog Box

4. Type any valid SQL Select statement based on the selected database type. Refer to the following sections of this chapter for the SQL Select syntax.

5. Choose OK when you have finished typing the SQL Select statement.

   The Database Source dialog box appears again.

6. Choose OK.

   The "Load the Database from the source?" prompt appears.

7. Choose Yes to load the database.

   - The SQL Select statement should include all the information you want. Although it should not be necessary, you can choose No to defer loading so that you can specify the data cache further. See *Chapter 3: Selecting the Database Source*.

Pasting an SQL Select Statement from a Q+E Query

If you have Pioneer Software System's Q+E program, you can paste in SQL Select statements without having to type.

❑ **To paste an SQL Select Statement from Q+E:**

1. In Q+E, perform a query.

2. Choose Edit Copy Special and turn on SQL Text.

   This copies the statement to the Clipboard.

3. Use the Windows Control menu to switch to Lens.

4. - To select a new database source, choose File New.
   - To switch database sources, choose Database Source.

   A dialog box appears, as shown in Figure 5-1.

5. Choose the appropriate database type and then choose the SQL button.

   A dialog box appears, as shown in Figure 5-2.

6. Select Paste.

   The statement on the Clipboard is pasted in the text box.

7. Use your edit keys to delete the reference to:

   dBASEFile|

8. Choose OK twice to close both dialog boxes.

   The "Load the Database from the source?" prompt appears.

9. Choose Yes to load the database.

   - The SQL Select statement should include all the information you want. If absolutely necessary, you can choose No to defer loading and further specify the data cache. See *Chapter 3: Selecting the Database Source*.

Database Systems

The rest of this chapter contains information on the following database systems:

- dBASE-compatible database files
- SQL Server databases
- Oracle databases
- Text files
- Excel worksheet files
- IBM DB2 databases
- NetWare SQL databases
- Paradox databases

dBASE-Compatible Database Files

The Q+E Database Library (QELIB) contains a driver that supports dBASE II, dBASE III, and dBASE IV-compatible database files. QELIB executes the SQL statements directly on dBASE-compatible files. You do not need to own the dBASE product in order to access these files.

Connection String

To connect to dBASE database files, the connection string used in qeConnect is:

```
"DRV=QELDBF"
```

QELDBF.DLL is the database driver used by QELIB to access dBASE-compatible database files. The file QELDBF.DLL must either be in your current directory, on your DOS PATH, or in the \WINDOWS\SYSTEM.

Create & Drop Table Statements

QELIB supports SQL statements to create and delete dBASE-compatible database files. The Create Table statement is used to create files and the Drop Table statement is used to delete files.

Create Table

A sample Create Table statement to create an employee database file is:

```
CREATE TABLE emp.dbf (last_name CHAR(20),
first_name CHAR(12), salary NUMERIC
(10,2), hire_date DATE)
```

The form of the Create Table statement is:

```
CREATE TABLE <filename> (<col_name>
<data_type>, ...)
```

The **filename** can be a simple file name (emp.dbf) or a full pathname (C:\QELIB\emp.dbf). If it is a simple file name, the file is created in the current working directory.

Column names (**col_name**) can contain up to 10 characters. You can use letters, numbers, or the underscore character in names. Names cannot contain blanks and the first character must be a letter.

Data_type is the specification of a column's data type. The possible data types in a Create Table statement are shown in Table 5-1.

Table 5-1 Data Types

Data Type	Description
Char or Character	Values can contain letters, numbers, or any of the punctuation keys on your keyboard. A length parameter is required, giving the maximum length of a character value that can be stored. The length limit is 254 characters. For example, CHAR(12).
Numeric or Number	Values can contain only numbers. Values can include a decimal point and a leading minus-sign. Two parameters give the maximum number of digits in the number and the number of digits right of the decimal point. The second parameter is optional. There is a limit of 19 total digits. For example, NUMERIC(10,2) declares a 10-digit number, 8 digits to the left of the decimal point and 2 digits to the right. NUMERIC (8) is equivalent to NUMERIC(8,0) and declares an 8- digit number with no digits to the right of the decimal point.
Float	Values are stored the same as numeric, but other programs can treat this field type differently. Float has the same parameters as numeric. For example, FLOAT(10,2) is the same as NUMERIC(10,2).

Date	Values contain date values. The time of day is not included. There are no parameters. For example, DATE.
Logical	Used for true/false or yes/no information. The possible values are the letters T, F, Y or N. There are no parameters. For example, LOGICAL.
Memo	Values contain long, multi-line textual data. There are no parameters. The value can be up to 65535 characters long.

Drop Table

A sample Drop Table statement to delete the employee database file is:

```
DROP TABLE emp.dbf
```

The form of the Drop Table statement is:

```
DROP TABLE <filename>
```

The **filename** can be a simple file name (emp.dbf) or a full pathname (C:\QELIB\emp.dbf). If it is a simple file name, the file must be in the current working directory.

Index Files

An Index is used to read records in sorted order and to improve performance when selecting records and joining files. A database file can have one or more indexes associated with it.

When you create an index, you specify the database field or field expression whose values are to be used to build the index. For example, an index built on the employee last_name field can be used to read employee records in last_name order without having to use an Order By clause.

Also, if a Where condition includes the indexed field, such as last_name = 'Woltman', QELIB finds the matching records much faster.

You can have more than one index associated with a database file. For example, you could have a last_name index and an emp_id index on an employee file, one to display the records in last name order, and one to rapidly look up employees given their employee id.

QELIB supports the dBASE II (.NDX), dBASE III (.NDX), and dBASE IV (.MDX) compatible index files. Each dBASE II and III index file contains one index. A dBASE IV index file can contain more than one index. In a dBASE IV index file, each index has a name called the tag name to identify the index. By default, dBASE IV index files have the same name as the database file, with the .MDX extension.

QELIB automatically opens the dBASE IV index file having the same name as the database file. If you are using dBASE II or III index files, you must open the files by including their names in SQL statements. You can have several index files open for the same database file. dBASE IV index files are more convenient and should be used whenever possible.

If you use the Insert, Update, and Delete statements, the indexes associated with the database file need to be updated. Otherwise, the index files will not match the records in the database file. To insure that the index files are maintained, you should always list the index files in the Insert, Update, and Delete statements.

Improving Record Selection Performance

Index files can improve the performance of Select, Update, and Delete statements. You can not notice this improvement with small files but it is significant for large files.

For indexes to improve the performance of selections, the index expression must match the selection condition exactly. For example, if you have created a dBASE IV index whose expression is last_name, the following Select statement uses the index:

```
SELECT * FROM emp.dbf WHERE last_name =
'Smith'
```

This Select statement does not use the index:

```
SELECT * FROM emp.dbf WHERE
UPPER(last_name) = 'SMITH'
```

The second statement does not use the index because the Where clause contains UPPER(last_name), which does not match the index expression last_name. If you plan to use the UPPER function in all your Select statements, then you should define an index whose expression is UPPER(last_name).

Also, to optimize a Where clause by using an index, you must not be using a different index to sort the records. If you have an index named emp_id as well as the last_name index, and your Select statement is:

```
SELECT * FROM emp.dbf (/emp_id) WHERE
last_name = 'Smith'
```

QELIB uses the emp_id index to return the records ordered by employee ID. The last_name index is not used.

Improving Join Performance

When joining database files, index files can greatly improve performance. For a Select statement such as:

```
SELECT * FROM dept.dbf, emp.dbf WHERE
dept.dept_id = emp.dept
```

QELIB uses an index on the dept field of the emp.dbf file. QELIB does not use any indexes on the dept.dbf file.

To improve join performance, you need an index on the join field of the second file in the From clause. If there is a third file in the From clause, QELIB also uses an index on the field in the third file that joins it to the second file.

Create & Drop Index Statements

QELIB supports SQL statements to create and delete indexes. The Create Index statement is used to create indexes and the Drop index statement is used to delete indexes.

Create Index

A sample Create Index statement to create an index on the employee ID field of the employee database file is:

dBASE IV format:

```
CREATE UNIQUE INDEX /emp_id on emp.dbf
(emp_id)
```

dBASE II or III format:

```
CREATE UNIQUE INDEX emp_id.ndx on emp.dbf
(emp_id)
```

The form of the Create Index statement is:
dBASE IV format:

```
CREATE [UNIQUE] INDEX [<index_file>]
/<tag_name> on <filename> (<index_expr>)
```

dBASE II or III format:

```
CREATE [UNIQUE] INDEX <index_file> on
<filename> (<index_expr>)
```

UNIQUE means that QELIB makes only one index entry for each unique value of the index expression. For example, each employee has a unique employee ID, so an index on employee ID should be unique. If you specify that an index is UNIQUE but the index values are not unique, no error is given and only one record having each value is stored in the index. For example, a unique index on last_name contains one entry for 'Smith'. If there are several employees with this last name, QELIB can find only one using the index.

The **index_file** is the name of the file to contain the index. For dBASE IV, index_file is optional. If you omit it, QELIB creates the index in the index file whose name is the same as the database file, with the .MDX extension. dBASE II or III require index_file since only one index is stored in each index file, and the file extension is .NDX.

The **/tag_name** is the name of the index. dBASE IV requires this to identify the indexes in an index file. Each index in an index file must have a different tag_name.

The **filename** is the name of the database file whose index is to be created.

The **index_expr** is the expression used to define the index. It is usually a single column name, such as last_name. However, you can use any valid dBASE expression. For example, use UPPER(last_name) to build an index on the upper cased last_names QELIB can perform case insensitive lookups. Use last_name+first_name to build a concatenated index on two fields so that QELIB retrieves the records in last_name, first_name order.

Drop Index

A sample Drop Index statement to delete an index on the employee ID field of the employee database file is:

dBASE IV format:

```
DROP INDEX /emp_id on emp.dbf (emp_id)
```

dBASE II or III format:

```
DROP INDEX emp_id.ndx
```

The form of the Drop Index statement is:

dBASE IV format:

```
DROP INDEX {<index_file> /<tag_name> |
/<tag_name> on <filename>}
```

dBASE II or III format:

```
DROP INDEX <index_file> [on <filename>]
```

The **index_file** is the name of the index file from which to delete the index. For dBASE IV you can either specify the index_file and tag_name, or the tag_name and the database filename. dBASE II or III requires index_file because only one index is stored in each index file, and the file extension is .NDX.

The **/tag_name** is the name of the index to be deleted. dBASE IV requires a tag_name to identify the index in an index file. Each index in an index file must have a different tag_name.

The **filename** is the name of the database file whose index is being deleted.

Select Statement

You use the SQL Select statement to specify the columns and records to be read.

For example.

```
SELECT *
FROM emp.dbf
```

The asterisk (*) following Select means to read all columns. The From clause is used to specify the file or table to be read. This Select statement selects all columns of all records in the emp.dbf database file

If you do not want all columns, specify the columns following Select:

```
SELECT first_name, last_name
FROM emp.dbf
```

This statement selects the first_name and last_name columns of all records in the emp.dbf database file.

To have the records returned in sorted order, add an Order By clause:

```
SELECT first_name, last_name
FROM emp.dbf
ORDER BY last_name
```

This statement selects the first_name and last_name columns of all records in the emp.dbf database file, and returns them sorted by last_name.

You can use column expressions:

```
SELECT first_name + last_name, salary *
1.1
FROM emp.dbf
```

This statement returns the first_name concatenated with the last_name as the first column, and the salary times 1.1 as the second column.

If you do not want to select all records, use a Where clause to specify the conditions that must be met for a record to be selected.

```
SELECT *
FROM emp.dbf
WHERE salary > 25000
```

This statement returns employee records for employees whose salary is greater than $25,000. All columns are selected.

Where clauses can become complex:

```
SELECT *
FROM emp.dbf
WHERE (salary > 25000 AND last_name =
'Smith')
     OR hire_date < {1/1/88}
```

This statement selects employees making more than $25,000 whose last name is Smith, or who were hired before Jan 1, 1988. Everyone hired before Jan 1, 1988 is selected, no matter what their salary or last_name is.

Select statements can return records from more than one file, this is called a join:

```
SELECT dept_name, last_name, first_name
FROM dept.dbf, emp.dbf
WHERE dept.dept_no = emp.dept
ORDER BY dept_name, last_name
```

This statement selects the department name column from the dept.dbf file, and the last_name and first_name columns from the emp.dbf file. The files to be joined are listed in the From clause. The Where clause indicates that the only combination of records to be selected are those where the department number field in the department record is equal to the department number field in the employee record.

The Order By clause causes the results to be returned ordered first by department name, then by last name within each department. The result is a list of all employees that includes the name of the department they work in, ordered by department.

Notice in the Where clause that the column names are prefixed by the file names. When you are joining files, this helps to clarify which columns are in which files. The file prefix is required if the same column name appears in both files. You can add the file prefix to all column names in the Select statement.

You can join a table to itself in some cases:

```
SELECT E1.last_name, E1.first_name,
E2.last_name, E2.first_name
FROM emp.dbf E1, emp.dbf E2
WHERE E1.mgr_id = E2.emp_id
ORDER BY E1.last_name
```

This statement returns employee names with their manager's name. The emp.dbf file is repeated twice in the From clause, and each file name is followed by a table alias (E1 and E2). A *table alias* is another name for the table or file that can be used as a prefix on column names. All column names must be prefixed by a table alias since every column appears in both files being joined.

General Form

The form of the Select statement supported by QELIB for dBASE files is:

```
SELECT  * | <col_expr>[, <col_expr>...]
FROM <file_spec>[, <file_spec>...]
[ WHERE <conditions> ]
[ ORDER BY <sort_expr>[, <sort_expr>...]
]
```

The Select and From clauses are required. The Where and Order By clauses are optional.

Select Clause

Follow SELECT with a list of column expressions you want to retrieve, or just an asterisk (*) to retrieve all columns.

The most common expression is simply a field name (e.g. last_name). More complex expressions can include mathematical operations or string manipulation.

You can list any number of column expressions as long as you separate them by commas. Field names can be prefixed with the file name or table alias (for example, emp.last_name). If more than one file is present in the From clause, you can use "SELECT file.*" to retrieve all fields from a file.

From Clause

Follow From with a list of file specifications (file_spec). File specifications have the form:

```
[pathname [options] [table_alias]]
```

The pathname can be a simple file name (emp.dbf) or a complete pathname (c:\qelib\emp.dbf). If a simple file name is given, the file must be in the current working directory. The .DBF extension is not required, QELIB automatically adds the extension if it is not present.

The table_alias is a name used to refer to this file in the rest of the Select statement. Database field names can be prefixed by the file name or table alias. Given the file specification:

```
FROM emp.dbf E
```

you can refer to the last_name field as E.last_name. You must use table aliases if the Select statement joins a table to itself. For example:

```
SELECT * FROM employee.dbf E, employee.dbf
F
WHERE E.mgr_id = F.emp_id
```

The options allow you to control whether deleted records are to be returned and which index files are to be opened and used for sorting.

The options specification is:

```
([COMPATIBILITY= {ANSI | DBASE}, ]
 [CHARSET= {ANSI | IBMPC}, ]
 [RECORDS= {DELETED | UNDELETED}, ]
 [index_spec])
```

The COMPATIBILITY option determines whether ANSI or dBASE IV-compatible SQL is to be used. The CHARSET option determines whether the database file uses the ANSI or IBM PC character set, and the RECORDS option determines if undeleted records are to be returned or deleted records. An example of an SQL Select statement using these options is:

```
SELECT * FROM emp.dbf (COMPATIBILITY=ANSI,
CHARSET=IBMPC, RECORDS=UNDELETED)
```

The index_spec controls the use of index files. For dBASE II and III files, the form is:

```
<index_filename> [/USE], ...
```

You specify the list of index filenames (.NDX extension) to be opened. In addition, you can specify that one of the index files is to be used to sort the records. If an index file is used to sort the records, the records are returned in the index order when no Order By clause is given.

For dBASE IV files, the index_spec can be:

```
[index_filename] [/tag_name], ...
```

You specify the list of index filenames (.MDX extension) to be opened. QELIB automatically opens the index file having the same name as the database file, if it exists. The **tag_name** is the name of the index within the index file to be used to sort the records. Here is an example using dBASE III index files:

```
SELECT * FROM emp.dbf (emphire.ndx,
empdept.ndx/USE)
```

Here is an example using a dBASE IV index:

```
SELECT * FROM emp.dbf (/depttag)
```

Because no index filename is specified, QELIB assumes that emp.MDX exists and contains an index named deptTAG.

Where Clause

The Where clause specifies the conditions that records must meet to be retrieved. The Where clause contains conditions in the form:

```
<expr1> <rel_operator> <expr2>
```

exp1 and **exp2** can be field names, constant values, or expressions. **rel_operator** is the relational operator that links the two expressions.

For example, the Select statement used to retrieve the names of employees that make at least $20,000 is:

```
SELECT last_name,first_name FROM emp.dbf
WHERE salary >= 20000
```

Order By Clause

The Order By clause indicates how the records are to be sorted. The form of the clause is:

```
<sort_expression> [DESC | ASC], ...
```

The **sort_expression** can be field names, expressions, or the number of the column expression to use. As an example, to sort by last_name, you could use either of the following Select statements:

```
SELECT emp_id, last_name, first_name FROM
emp.dbf ORDER BY last_name

SELECT emp_id, last_name, first_name FROM
emp.dbf ORDER BY 2
```

In the second example, last_name is the second column expression following Select, so Order By 2 sorts by last_name.

SQL Expressions

Expressions are used in the column expressions, Where clauses, and Order By clauses of SQL Select statements. They are also used to define indexes.

Expressions allow you to use mathematical operations as well as character string and date manipulation, to form complex database queries.

QELIB's dBASE support provides a set of operators and functions you can use in expressions. The values in expressions can come from fields of the database records or constant values.

Constants

Constants are values that do not change. For example, in the expression PRICE * 1.05, the value 1.05 is a constant.

You must enclose character constants in pairs of single (') or double quotes ("). To include a single quote in a character constant enclosed by single quotes, use two single quotes together (for example, 'Don''t'). Similarly, if the constant is enclosed by double quotes, use two double quotes to include one.

You must enclose date constants in braces ({}) (for example, {01/30/89}). Date constants must be in the form MM/DD/YY. The two logical constants are .T. for true and .F. for false.

Numeric Expressions

You can include the following operators in numeric expressions:

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
^	Exponentiation

The following chart shows examples of numeric expressions, where salary is 20000:

Example	Resulting value
salary + 10000	30000
salary * 1.1	22000
2 ** 3	8

You can precede numeric expressions with a unary plus (+) or minus (-). For example -(salary * 1.1) is -22000.

Character Expressions

Character expressions can include the following operators:

Operator	Meaning
+	Concatenation keeping trailing blanks.
-	Concatenation moving trailing blanks to the end.

The following chart shows examples of character expressions. In the examples, last_name is 'BENNETT ' and first_name is 'TYLER ':

Example	Resulting value
first_name+last_name	'TYLER BENNETT '
first_name-last_name	'TYLERBENNETT '
first_name-(' '+last_name)	'TYLER BENNETT '

Date Expressions

You can include these operators in date expressions.

Operator	Meaning
+	Add a number of days to a date to produce a new date.
-	The number of days between two dates, or subtract a number of days from a date to produce a new date.

The following shows examples of date expressions. Assume that hire_date is {01/30/90}.

Example	Resulting value
hire_date + 5	{02/04/90}
hire_date - {01/01/90}	29
hire_date - 10	{01/20/90}

Relational Operators

The relational operators (**rel_operator**) separating the two expressions can be:

Operator	Meaning
=	Equal
<>	Not Equal
!=	Not Equal
#	Not Equal
>	Greater Than
>=	Greater Than or Equal
!<	Not Less Than (same as Greater Than or Equal)
<	Less Than
<=	Less Than or Equal
!>	Not Greater Than (same as Less Than or Equal)
LIKE	Matching a pattern
NOT LIKE	Not matching a pattern
*=	Outer Join, Matches NULL for second value
=*	Outer Join, Matches NULL for first value
IS NULL	Equal to NULL
IS NOT NULL	Not Equal to NULL
BETWEEN	Values between a lower and upper bound

Examples

```
salary <= 40000
dept = 'D101'
hire_date > {01/30/89}
salary + commission  >= 50000
last_name LIKE 'Be%'
salary IS NULL
salary BETWEEN 10000 AND 20000
```

Logical Operators

Two or more conditions can be combined to form more complex criteria. They must be related by AND or OR. For example:

```
salary = 40000 AND exempt = .T.
```

Use the logical NOT operator to reverse the meaning. For example:

```
NOT (salary = 40000 AND exempt = .T.)
```

Operator Precedence

As expressions become more complex, the order in which they are evaluated is important. The following table shows the order of evaluating operators. Operators in the first line are evaluated first, then the second line and so on. Operators in the same line are evaluated left to right.

Precedence	Operators
1	Unary -, Unary +
2	**, ^
3	*, /
4	+, - (between numbers, dates, or character strings)
5	=, <>, !=, #, <, <=, !<, >, >=, !>, *=, =*, LIKE, NOT LIKE, IS NULL, IS NOT NULL, BETWEEN,
6	NOT
7	AND
8	OR

The following example shows the importance of precedence:

```
WHERE salary > 40000 OR
      hire_date > {01/01/89} AND
      dept = 'D101'
```

Because AND is evaluated first, this query retrieves employees in department D101 hired after Jan 1, 1989, as well as every employee making more than $40,000, no matter what department or hire date.

To force the clause to be evaluated in a different order, use parentheses to enclose the conditions to be evaluated first. For example:

```
WHERE (salary > 40000 OR hire_date >
   {01/01/89}) AND dept = 'D101'
```

retrieves employees in department D101 that either make more than $40,000 or were hired after Jan 1, 1989.

Functions

QELIB supports a number of dBASE functions that you can use in expressions. In the following charts, the functions are grouped according to the type of result they return.

Functions that Return Character Strings

Function	Description
RTRIM	Removes trailing blanks from a string. RTRIM('ABC ') returns 'ABC'.
TRIM	Same as RTRIM.TRIM('ABC ') returns 'ABC'.
LTRIM	Removes leading blanks from a string. LTRIM(' ABC') returns 'ABC'.
UPPER	Changes each letter of a string to uppercase. UPPER('Rappl') returns 'RAPPL'.
LOWER	Changes each letter of a string to lowercase. LOWER('Rappl') returns 'rappl'.

LEFT	Returns left-most characters of a string.LEFT('Woltman',3) returns 'Wol'.
RIGHT	Returns right-most characters of a string. RIGHT('Woltman',4) returns 'tman'.
SUBSTR	Returns a substring of a string. Parameters are the string, the first character to extract, and the number of characters to extract (optional). SUBSTR('Holcomb',2,3) returns 'olc'. SUBSTR('Holcomb',2) returns 'olcomb'.
SPACE	Generates a string of blanks. SPACE(5) returns ' '.
DTOC	Converts a date to a character string. An optional second parameter determines the format of the result: 0, the default returns MM/DD/YY. 1 returns DD/MM/YY. 2 returns YY/MM/DD. 10 returns MM/DD/YYYY. 11 returns DD/MM/YYYY. 12 returns YYYY/MM/DD. An optional third parameter specifies the date separator character. If not specified, a slash (/) is used: DTOC({01/30/89}) returns '01/30/89'. DTOC({01/30/89}, 0) returns '01/30/89'. DTOC({01/30/89}, 1) returns '30/01/89'. DTOC({01/30/89}, 2,'-') returns '89-01-30'.
DTOS	Converts a date to a character string using the format YYYYMMDD. DTOS({01/23/90}) returns '19900123'.
IIF	Returns one of two values. Parameters are a logical expression, the true value, and the false value. If the logical expression evaluates to

true, the function returns a true value. Otherwise it returns a false value.

IIF(salary>20000,"BIG","SMALL") returns "BIG" if the salary is greater than 20000. If not, it returns "SMALL".

STR	Converts a number to a character string. Parameters are the number, the total number of output characters (including the decimal point), and optionally the number of digits to the right of the decimal point. STR(12.34567,4) returns '12'. STR(12.34567,4,1) returns '12.3'. STR(12.34567,6,3) returns '12.346'.

Functions that Return Numbers

Function	Description
MOD	Divides two numbers and returns the remainder of the division. MOD(10,3) returns 1.
LEN	Returns the length of a string. LEN('ABC') returns 3.
MONTH	Returns the month part of a date. MONTH({01/30/89}) returns 1.
DAY	Returns the day part of a date. DAY({01/30/89}) returns 30.
VAL	Converts a character string to a number. The parameter is a character string. If the parameter is not a valid number, a zero is returned. VAL("123") returns the number 123.
YEAR	Returns the year part of a date. YEAR({01/30/89}) returns 1989.

Functions that Return Dates

Function	Description
DATE	Returns today's date. DATE() returns today's date.
CTOD	Converts a character string to a date. An optional second parameter specifies the format of the character string: 0 (the default) returns MM/DD/YY, 1 returns DD/MM/YY, and 2 returns YY/MM/DD. CTOD('01/30/89') returns {01/30/89}. CTOD('30/01/89',1) returns {01/30/89}.

Function Examples

The following examples use the dBASE functions.

Retrieve all employees that have been with the company at least 90 days:

```
SELECT first_name,last_name FROM emp
   WHERE DATE() - hire_date  >= 90
```

Retrieve all employees hired in January of this year or in January of last year:

```
SELECT first_name,last_name FROM emp
   WHERE MONTH(hire_date) = 1
   AND (YEAR(hire_date) = YEAR(DATE())
   OR YEAR(hire_date) = YEAR(DATE()) - 1)
```

ANSI SQL Compatibility

By default, QELIB supports SQL as defined by dBASE IV. There are several differences between the dBASE IV product and standard SQL as defined by ANSI (American National Standards Institute). If you prefer to use ANSI SQL rather than dBASE IV SQL, use the file_spec options described earlier, or use the QELSETUP program to change the default.

Character Fields

If you compare a character field to a character constant in a Where clause, (for example, last_name = 'S'), dBASE SQL returns every record whose last name begins with 'S'. ANSI SQL only returns those records with a last name that is exactly 'S'.

If you use the UPPER or LOWER functions, dBASE SQL does not change the case of international characters (for example, umlauts and accents).

NULL Values

A record's field has a Null value if it has no value. ANSI SQL has special rules for Null values, but dBASE SQL does not support them. If you choose ANSI compatibility, QELIB treats blank field values as Null.

The following chart shows how blank values are sorted depending on your choice of ANSI or dBASE SQL:

Data type	dBASE	ANSI
number	Sorted as 0	Placed at front
date	Sorted as largest date	Placed at front
logical	Sorted as .F.	Placed at front
character	Sorted as blanks	Placed at front

If a record has a blank field, ANSI and dBASE SQL differ in how the record is selected by a Where clause. For a field named X, the following conditions select records with blank values:

Data Type	dBASE	ANSI
number	X = 0	X IS NULL
date	X = { / / }	X IS NULL
logical	X = .F.	X IS NULL
character	X = ' '	X IS NULL

Finally, dBASE and ANSI SQL generate different results when you have expressions that contain Null values.

For numeric expressions involving a Null value, (for example, AMOUNT * 1.1 when AMOUNT is blank), dBASE treats blanks as zero. ANSI always returns the Null value.

For date expressions involving a Null value, ANSI always returns the Null value. The following table shows how dBASE treats Null values. In this example, hire_date is blank:

Expression	dBASE	ANSI
hire_date - {01/01/89}	0	NULL
hire_date + 10	blank	NULL
hire_date - 10	blank	NULL

For logical expressions, dBASE treats blank values as .F., and ANSI treats blank values as Null.

The following table gives the rules that ANSI applies when using AND and OR with Null expressions.

Expression 1	Operator	Expression 2	End Result
TRUE	AND	NULL	NULL
FALSE	AND	NULL	FALSE
TRUE	OR	NULL	TRUE
FALSE	OR	NULL	NULL
NULL	AND/OR	NULL	NULL

Insert Statement

The SQL Insert statement is used to add new records to a database file. For example:

```
INSERT INTO emp.dbf (last_name,
   first_name, emp_id, salary, hire_date)
   VALUES ('Smith', 'John', 'E22345',
   27500, {4/6/91})
```

Each Insert statement adds one record to the database file. In this case a record has been added to the employee database file, emp.dbf. Values are specified for four columns. The remaining columns in the file are assigned a blank value, meaning Null. Note that character values must be enclosed in quotation marks and dates must be enclosed in braces {}.

The form of the Insert statement supported for dBASE files is:

```
INSERT INTO <filename>
   [(<index_file>,...)] [(<col_name>,...)]
   VALUES (<expr>, ...)]
```

The list of indexes and the column names are optional.

The **filename** is the name of the database file QELIB adds the record to.

The **index_file** list is the list of index files to be updated when the record is added. You should list every index file that has been created for the database file. If you have a dBASE IV index file that has the same name as the database file (emp.mdx in this case), it does not have to be listed because QELIB opens it automatically.

The **col_name** list is an optional list of column names giving the name and order of the columns whose values are specified in the Values clause. If you omit col_name, the value expressions (**expr**) must be in the order the columns are defined for the file.

The **expr** list is the list of expressions giving the values for the columns of the new record. Usually, the expressions are constant values for the columns. If you specify the col_name list, the values must be in the order the columns are listed. If the col_name list is omitted, the values must be in the order of the columns in the file. Character string values must be enclosed with single or double quote characters, date values must be enclosed by braces {}, and logical values must be enclosed by periods (e.g., .T. or .F.).

Update Statement

The SQL Update statement changes records in a database file. This is an of an Update statement on the employee file:

```
UPDATE emp.dbf SET salary=32000,
   exempt=.T. WHERE emp_id = 'E10001'
```

The Update statement changes every record that meets the conditions in the Where clause. In this case, the salary and exempt status has been changed for all employees having the employee ID E10001. Since employee ID's are unique in the employee file, one record is updated.

The Update statement supported for dBASE files:

```
UPDATE <filename> [(<index_file>,...)]
   SET <col_name> = <expr>, ...
   [ WHERE <conditions> ]
```

The **filename** is the name of the database file to be updated.

The optional **index_file** is a list of index files to be updated when the record is updated. It improves the performance of the Update statement. You should list every index file that has been created for the database file. If you have a dBASE IV index file with the same name as the database file, QELIB opens it automatically.

col_name is the name of a column whose value is to be changed. Several columns can be changed in one statement.

The **expr** is the new column value. Usually, the expression is a constant value. Character string values must be enclosed with single or double quotes, date values must be enclosed by braces {}, and logical values must be enclosed by periods.

The optional Where clause is any valid clause for dBASE files. See "Select statements" to determine the records to be updated.

Delete Statement

The SQL Delete statement deletes records from a database file. An example of a Delete statement on the employee file is:

```
DELETE FROM emp.dbf
    WHERE emp_id = 'E10001'
```

Each Delete statement removes every record that meets the conditions in the Where clause. In this case, every record having the employee id E10001 is deleted. Because employee ID's are unique in the employee file, one record is deleted.

When records are deleted from a dBASE file, they are not removed from the file. Instead, they are marked as having been deleted.

The form of the Delete statement supported for dBASE files is:

```
DELETE FROM <filename> [(<index_file>,
    ...)] [ WHERE <conditions> ]
```

The list of indexes and the Where clause are optional.

The **filename** is the name of the database file whose records are to be deleted.

The **index_file** list provides the index files to be used to improve the performance of the Delete statement. Index files are not maintained by Delete statements since the records are not physically removed from the file. If you have a dBASE IV index file that has the same name as the database file (emp.mdx in this case), QELIB will open it automatically.

The Where clause is any valid clause for dBASE files. See "Select Statements" to determines which records are to be Deleted.

Commit and Rollback

The dBASE driver immediately executes Insert, Update, and Delete statements on the database files. The changes are automatically committed when the SQL statement is executed. qeBeginTran, qeCommit, and qeRollback cannot be used with dBASE files.

Data Types

The following table shows how the dBASE data types are mapped to the eight standard QELIB data types returned by qeColType, and to the underlying database types returned by qeColDBType:

dBASE	qeColType	qeColDBType
Character	Fixed length character string, type 1	1
Date	Date-time, type 8	111
Float	Decimal number, type 3	3
Logical	Integer, type 5	110
Memo	Variable length character string, type 2	100
Numeric	Decimal number, type 3	3
Numeric Expr	Double float, type 7	7
Char Expr	Variable length character string, type 2	100

Numeric Expr and Char Expr are used for expressions. For example:

```
SELECT salary*1.1, last_name+first_name
FROM emp.dbf
```

salary*1.1 is a numeric expression, and last_name+first_name is a character expression.

SQL Server Databases

The Q+E Database Library (QELIB) contains a driver that supports the SQL Server database system available from Sybase® and Microsoft.

QELIB executes the SQL statements by sending them to the SQL Server database system. Any SQL statement supported by the database system can be executed from QELIB. See your SQL Server documentation for more information.

Connection String

The qeConnect function is used to connect to SQL Server databases. Connecting to SQL Server logs you onto a server computer and opens a connection on which you can execute SQL statements. The connection string sent to qeConnect has the form:

<attribute>=<value>[;<attribute>=<value>...]

The attributes used by SQL Server are:

Attribute	Description
DRV	The name of the driver for the database system, always QESS for SQL Server.
UID	Login ID, these are case sensitive.
PWD	Password, also case sensitive.
SRVR	Name of the server computer containing the SQL Server database tables you wish to access.

An example of a connection string for SQL Server is:

```
"DRV=QESS;SRVR=PION1;UID=sa;PWD=XYZZY"
```

The database driver used by QELIB to access the SQL Server database systems is named QESS.DLL or QELSS.DLL. The DLL must either be in your current directory, in your DOS PATH, or in the \WINDOWS\SYSTEM directory.

SQL Statements

Once you have opened a connection, you can execute SQL statements on the connection by calling qeExecSQL. You can execute any statement acceptable by SQL Server.

With SQL Server, you are limited to executing only one SQL statement per connection. You cannot have simultaneous statements active on one connection.

To execute a second SQL statement while the first one is active, you can open a second connection by calling qeConnect a second time. You can have several connections open to SQL Server simultaneously, each executing one SQL statement.

Commit and Rollback

The qeCommit and qeRollback functions cause the corresponding SQL Server functions to be called.

Data Types

The following table shows how the SQL Server data types are mapped to the eight standard QELIB data types returned by qeColType, and to the underlying database types returned by qeColDBType:

SQL Server	qeColType	qeColDBType
Char	Variable length character string, type 2	2
Varchar	Variable length character string, type 2	2
Text	Variable length character string, type 2	100
Int	Long integer, type 4	4
SmallInt	Integer, type 5	5
TinyInt	Integer, type 5	109
Bit	Integer, type 5	110
Float	Double float, type 7	7
Money	Decimal number, type 3	116
DateTime	Date-Time, type 8	8
TimeStamp	Variable length character string, type 2	114
	if field name is "timestamp", otherwise	101
Binary	Variable length character string, type 2	101
VarBinary	Variable length character string, type 2	101
Image	Variable length character string, type 2	103

Oracle Databases

The Q+E Database Library (QELIB) contains a driver that supports the Oracle database system.

QELIB executes the SQL statements by sending them to the Oracle database system. Any SQL statement supported by the database system can be executed from QELIB. See your Oracle documentation for more information.

Connection String

The qeConnect function is used to connect to Oracle databases. Connecting to Oracle logs you onto a server computer and opens a connection on which you can execute SQL statements. The connection string sent to qeConnect has the form:

<attribute>=<value>[;<attribute>=<value>...]

The attributes used by Oracle are:

Attribute	Description
DRV	The name of the driver for the database system, always QEORA for Oracle.
UID	Username.
PWD	Password.
SRVR	The SQL*Net connect string designating the server computer and database to be accessed.

The most difficult attribute value to specify is the SRVR value. The information required varies depending on the SQL*Net driver you are using. See your SQL*Net documentation for more information.

The database driver used by QELIB to access the Oracle database systems is named QEORA.DLL or QELORA.DLL.

The DLL must either be in your current directory, in your DOS PATH, or in the \WINDOWS\SYSTEM directory.

SQL Statements

Once you have opened a connection, you can execute SQL statements on the connection by calling qeExecSQL. You can execute any statement acceptable by Oracle.

Commit and Rollback

The qeCommit and qeRollback functions cause the corresponding Oracle functions to be called.

Data Types

The following table shows how the Oracle data types are mapped to the eight standard QELIB data types returned by qeColType, and to the underlying database types returned by qeColDBType.

Oracle	qeColType	qeColDBType
Char	Variable length character string, type 2	2
Date	Date-time, type 8	8
Long	Variable length character string, type 2	100
Long Raw	Variable length character string, type 2	103
Number	Double float, type 7	107
Raw	Variable length character string, type 2	101
ROWID	Fixed length character string, type 1	114

Text Files

The Q+E Database Library (QELIB) contains a driver that supports ASCII text files. QELIB executes the SQL statements directly on the Text files. QELIB is read-only for text files, you cannot execute Insert, Update, or Delete statements on these files.

Text files are readable, ASCII files. Text files can be printed directly or edited with text editors or word processors, since none of the data is stored in a binary format.

Some of the common formats for text database files are:

Comma-Separated Values

Commas are used to separate column values, and each line is a separate record. Each column value can be a different length, since commas are used to separate values. These files often have the .CSV extension.

Comma- and Tab-separated files are called character-delimited files, since values are separated by a special character. QELIB supports character-delimited files where the separator character is any printable character or the Tab character.

Tab-Separated Values

Tabs are used to separate column values, and each line is a separate record. Column values can vary in length, since tabs separate values.

Fixed-Format Files

No special character is used to separate column values. Instead, values start at the same position in each line and have the same length in each line. The values appear in fixed columns if you display the file. Each line is a separate record.

Connection String

To connect to text files, the connection string used in qeConnect is:

```
"DRV=QETXT"
```

QETXT.DLL or QELTXT.DLL is the database driver used by QELIB to access text files. The DLL must either be in your current directory, in your DOS PATH, or in the \WINDOWS\SYSTEM directory.

Select Statement

SQL for text files supports the same functions and expressions as SQL for dBASE files. See "dBASE-Compatible Database Files" for a complete description of SQL for dBASE files.

The one clause that contains information unique to text files is the From clause. The From clause for text files is described below.

Column Names

Text files can contain column names in the first line of the file. Whether the first line of the file contains column names is determined by the HEADERLINE option in the From clause, described later. If the first line contains column names, then the column names in the text file must be used as the column names in the Select statement.

If the first line does not contain column names, then the column names will be FIELD_1, FIELD_2, etc.

From Clause

Follow From with a list of file specifications. File specifications have the form:

```
pathname [options] [table_alias]
```

The **table_alias** is a name used to refer to this file in the rest of the Select statement. Database column names can be prefixed by the table alias. Given the file specification:

```
FROM emp.CSV E
```

You can refer to the last_name column as E.last_name. Table aliases must be used if the Select statement joins a table to itself. For example:

```
SELECT * FROM empLOYEE.CSV E, emp.CSV F
WHERE E.MANAGER_ID = F.emp_id
```

The options allow you to control the File Open options and include the Layout Parse Record string. The options specification for character-delimited files is:

```
([HEADERLINE= {0 | 1}, ]
 [CHARSET= {ANSI | IBMPC}, ]
 [DELIMITER= {TAB | 'char'}, ]
 [data_type_spec])
```

The options specification for fixed-format files is:

```
([HEADERLINE= {0 | 1}, ]
 [CHARSET= {ANSI | IBMPC}, ]
 [PARSE= parse_string, ]
 [data_type_spec])
```

The HEADERLINE option determines whether the first line in the file contains column names. The CHARSET option determines whether the database file uses the ANSI or IBM PC character set. Each of these is optional. An example of an SQL Select statement using these options is:

```
SELECT * FROM emp.CSV (HEADERLINE=1,
CHARSET=IBMPC)
```

For character-delimited files, the DELIMITER option determines the character used to separate values. Use TAB for tab-separated values and enclose any other character in quotation marks.

For fixed-format files, the PARSE option gives the parse string to be used to determine the position and length of each column in the file. By default, each line is treated as the value in one column. QELIB does not know where each column begins and ends in the line.

The parse string is a sample line in which you have entered a "[" in front of each column and a "]" at the end of each column. These bracket pairs "[ ]" enclose each column, indicating to QELIB the position and length of each column in the record.

For the sample file EMP.TXT, the correct parse line is:

```
[FIRST_NA][last_name ][emp_id][hire_date
][salary   ][dept][E]
```

The **data_type_spec** lets you specify the data type of each column in the file. When QELIB opens a fixed-format text file, it assumes that all columns contain character values.

For character-delimited files, QELIB guesses at the maximum width of the values in each column by scanning records.

If your text file contains numbers or dates as well as character values, QELIB needs this information. QELIB uses the column data types to properly execute the Where clause and Order By clauses, and to return the data in the appropriate data type.

The format of the data_type_spec is:

```
{[CHAR(width)  |
NUM(width,[decimal_digits])  |
DATE(format_string)],...}
```

For each column, you specify whether the data type is Character, Number, or Date.

If the type is Character, enter the maximum number of characters for the columns' values as the width.

If the type is Number, enter the maximum number of digits as the width. Also, enter the number of digits to the right of the decimal point as the decimal_digits.

If the type is Date, enter the appropriate format_string which shows the format of the date values in the file.

Commit and Rollback

The text driver does not support Insert, Update, and Delete statements. qeBeginTran, qeCommit, and qeRollback functions cannot be used.

Data Types

The following table shows how the text file data types are mapped to the eight standard QELIB data types returned by qeColType, and to the underlying database types returned by qeColDBType:

Text File	qeColType	qeColDBType
Character	Varying length character string, type 2	2
Date	Date-time, type 8	111
Numeric	Decimal number, type 3	3
Numeric Expr	Double float, type 7	7

Numeric Expr is used for expressions. For example:

```
SELECT salary*1.1 FROM emp.txt
(HEADERLINE=1, CHARSET=IBMPC, PARSE=
"first_nalast_name emp_idhire_date [salary
]depte",NUMBER(10,2))
```

Excel Files

The Q+E Database Library (QELIB) has a driver that supports Excel XLS files. You can open any Excel worksheet file that contains a database section. QELIB accesses the rows and columns from the database section. The Excel Data menu's Set Database command creates a database section in a worksheet. QELIB executes the SQL statements directly on the Excel files. QELIB is read-only for Excel files, so you cannot execute Insert, Update, or Delete statements on them.

Connection String

To connect to Excel files, use this connection string in qeConnect:

```
"DRV=QEXLS"
```

QEXLS.DLL or QELXLS.DLL is the database driver used by QELIB to access Excel files. The DLL must be in the current directory, the DOS PATH, or the \WINDOWS\ SYSTEM directory.

Select Statement

SQL for Excel files supports the same functions and expressions as SQL for dBASE files. See "dBASE-Compatible Database Files" for details.

Column Names

Excel files contain column names in the first row of the database section which you must use as the column names in the Select statement.

From Clause

The From clause contains information unique to Excel files and has the form:

```
pathname [data_type_spec] [table_alias]
```

table_alias is a name for referring to this file in the rest of the Select statement.

Database field names can be prefixed by the table alias. Given:

```
FROM emp.xls E
```

you can refer to the last_name field as E.last_name. You must use a table alias if the Select statement joins a table to itself. For example:

```
SELECT * FROM emp.xls E, emp.xls F
WHERE E.manager_id = F.emp_id
```

The **data_type_spec** specifies the data type of each field in the file. The Excel database driver guesses the data types if you do not give a specification. QELIB needs the column data types to properly execute the Where and Order by clauses and to return the data in the appropriate data type. The data_type_spec format is:

```
(CHAR(width) | FLOAT | DATE | INTEGER |
MONEY | LOGICAL,...)
```

For each column, specify if the data type is Character, Floating point number, Date, Integer, Money, or Logical. If the type is Character, enter the maximum number of characters for the columns' values as the width.

Commit and Rollback

The Excel driver does not support Insert, Update, and Delete statements. You cannot use qeBeginTran, qeCommit, and qeRollback functions.

Data Types

The Excel file data types are mapped to the eight standard QELIB data types returned by qeColType, and to the underlying database types returned by qeColDBType.

Excel Types	qeColType	qeColDBType
Character	Varying length character string, type 2	2
Date	Date-time, type 8	8
Float	Double float, type 7	7
Integer	Long Integer, type 4	4
Logical	Integer, type 5	110
Money	Decimal number, type 3	3

IBM DB2 Databases

The Q+E Database Library contains a driver that supports the IBM Database 2 (DB2) database system. QELIB uses the MicroDecision Ware (MDI) Database Gateway to provide remote access to DB2 databases.

QELIB executes the SQL statements by sending them to DB2. Any SQL statement supported by the database system can be executed from QELIB. See your DB2 documentation for more information.

Installation Notes

The QELIB Setup program is used to install the DB2 driver. Before you can use the DB2 driver with QELIB, follow these steps:

- Install the MicroDecision Ware Database Gateway.
- Test your connection to the Database Gateway by using either ISQL or SAF, as suggested in the *Database Gateway for DB2 Reference and Installation Guide*.
- Run the QELIB SETUP program to install QELIB and the DB2 driver.

Connection String

The qeConnect function is used to connect to DB2. Connecting to DB2 logs you on and opens a connection on which you can execute SQL statements.

The connection string sent to qeConnect has the form:

<attribute>=<value>[;<attribute>=<value>...]

The attributes used by DB2 are:

Attribute	Description
DRV	The name of the driver for the database system, always QEDB2 for DB2.
SRVR	Name of the server computer running the MDI Database Gateway.
UID	Authorization ID.
PWD	Password.

An example of a connection string for DB2 is:

```
"DRV=QEDB2;SRVR=PION1;UID=PIONEER;PWD=XYZZ
Y"
```

The database driver used by QELIB to access DB2 is named QLDB2.DLL.

The DLL must either be in your current directory, on your DOS PATH, or in the \WINDOWS\SYSTEM directory.

SQL Statements

Once you have opened a connection, you can execute SQL statements on the connection by calling qeExecSQL. You can execute any statement acceptable by DB2.

With DB2, you are limited to executing only one SQL statement per connection. You cannot have simultaneous statements active on one connection.

To execute a second SQL statement while the first one is active, you can open a second connection by calling qeConnect a second time. You can have several connections open to DB2 simultaneously, each executing one SQL statement.

Commit and Rollback

The qeCommit and qeRollback functions cause the corresponding DB2 functions to be called.

Data Types

The following table shows how the DB2 data types are mapped to the eight standard QELIB data types returned by qeColType, and to the underlying database types returned by qeColDBType:

Database Mgr	qeColType	qeColDBType
Char	Fixed length character string, type 1	1
Varchar	Variable length character string, type 2	2
Integer	Long integer, type 4	4
SmallInt	Integer, type 5 5	
Real	Single float, type 7	7
Float	Double float, type 8	8
Decimal	Decimal number, type 3	3
Date	Date-Time, type 8	111
Time	Date-Time, type 8	112
TimeStamp	Date-Time, type 8	8

Note

The Char for Bit Data, Varchar for Bit Data, Long Varchar, Graphic, Vargraphic, and Long Vargraphic DB2 data types are not supported by QELIB. If you create DB2 tables using any of these types, QELIB won't be able to read the records contained in those tables.

NetWare SQL Databases

The QELIB Database Library (QELIB) contains a driver that supports the NetWare SQL database system.

QELIB executes the SQL statements by sending them to the NetWare SQL database system. Any SQL statement supported by the database system can be executed from QELIB. See your NetWare SQL documentation for more information.

NetWare SQL stores records in XQL databases. An XQL database contains data files that contain your records and data dictionary files that describe the database itself. The data files are Btrieve tables. The data dictionary files are special Btrieve tables that contain complete descriptions of the data files, views, and fields in your database.

If a database administrator establishes security for an XQL database, then the data dictionary files contain lists of users who have access to the database and are used to enforce access rights to specific tables within the database.

Btrieve files must be incorporated into an XQL database before they can be accessed by QELIB using the NetWare SQL driver.

System Requirements

To access an XQL database across a Novell network, you must be using the following software:

- NetWare SQL 2.11 or higher must be running on the file server.
- NSREQ.EXE 2.12a or higher must be running on the client workstation. (These files come with the NetWare SQL product from Novell.)
- Windows 3.00a or higher.

Installation Notes

The QELIB Setup program is used to install the NetWare SQL driver for Windows. Before you can use the NetWare SQL driver with QELIB, you must follow these steps:

- Run the Windows setup program and choose Novell for your network. The Windows setup program modifies your SYSTEM.INI file to provide Novell network access.

- Edit your WIN.INI file. Under the [Windows] section, include the line: "load=nwpopup.exe". (NWPOPUP.EXE come with the NetWare SQL product from Novell.) Add the following sections to the WIN.INI file:

```
[nsreqDPMI]
datalength=4096
views=8
tasks=2
[btrieve]
options=/m:32 /f:20 /p:1024
[brequest]
options=/r:20 /s:20 /d:532 /w:2
```

The WBTRCALL.DLL, WXQLCALL.DLL, and QEXQL.DLL dynamic link libraries must be located in a directory on your DOS PATH, or in the \WINDOWS\SYSTEM directory. (WBTRCALLS.DLL and WXQLCALLS.DLL come with the NetWare SQL product from Novell. QEXQL.DLL comes with QELIB.)

☞ Note

Before running Windows, you must log on to your Novell network and run NSREQ.EXE. (NSREQ.EXE comes with the NetWare SQL product from Novell.

Granting Access Rights

If security has been enabled on an XQL database, then the database administrator must grant each user SELECT access rights against some system tables in order for QELIB to function properly. If these access rights are not granted, QELIB will not display any tables in the File Open dialog box.

To grant the necessary access rights, an administrator can run XQLI.EXE and execute this:

```
GRANT SELECT ON X$File, X$Rights, X$User
TO PUBLIC
```

If views exist, the database administrator should also execute this statement:

```
GRANT SELECT ON X$View TO PUBLIC
```

If procedures exist, the database administrator should also execute this:

```
GRANT SELECT ON X$Proc TO PUBLIC
```

Connection String

The qeConnect function is used to connect to NetWare SQL databases. Connecting to NetWare SQL logs you onto a server computer and opens a connection on which you can execute SQL statements. The connection string sent to qeConnect has the form:

<attribute>=<value>[;<attribute>=<value>...]

Attributes Used by NetWare SQL

Attribute	Description
DRV	The name of the driver for the database system, always QEXQL for NetWare SQL.
FILES	The full pathname of the directory containing the NetWare SQL data files.
DICT	The full pathname of the directory containing the NetWare SQL dictionary files.
UID	Username (optional).
PWD	Password (optional).

The database driver used by QELIB to access the NetWare SQL database systems is named QLXQL.DLL. The DLL must be in your current directory, on your DOS PATH, or in the \WINDOWS\SYSTEM directory.

The UID and PWD are only required if security has been enabled in NetWare SQL.

SQL Statements

Once you have opened a connection, you can execute SQL statements on the connection by calling qeExecSQL. You can execute any statement acceptable by NetWare SQL.

Commit and Rollback

The qeCommit and qeRollback functions cause the corresponding NetWare SQL functions to be called.

Data Types

This table shows how the NetWare SQL data types are mapped to the eight standard QELIB data types returned by qeColType, and to the underlying database types returned by qeColDBType

NetWare SQL	qeColType	qeColDBType
Char	Fixed length character string, type 1	1
LString	Variable length character string, type 2	2
ZString	Variable length character string, type 2	1001
Int (1-byte)	Integer, type 5	109
Int (2-byte)	Integer, type 5	5
Int (4-byte)	Long integer, type 4	4
Float (4-byte)	Single Float, type 6	6
Float (8-byte)	Double Float, type 7	7
Date	Date-time, type 8	111
Time	Date-time, type 8	112
Decimal	Decimal number, type 3	3
Money	Decimal number, type 3	116
Logical	Integer, type 5	110
BFloat (4-byte)	Single Float, type 6	1004
BFloat (8-byte)	Double Float, type 7	1005
Numeric	Decimal number, type 3	1006
Autoinc (2-byte)	Integer, type 5	1007
Autoinc (4-byte)	Long integer, type 4	1008
Bit	Integer, type 5	1003
Note	Variable length character string, type 2	100
LVar	Variable length character string, type 2	1009

Paradox Database Files

The Q+E Database Library (QELIB) contains a driver that supports Paradox database files.

QELIB accesses Paradox files through Borland's Paradox Engine version 2.0 or later. You must install the Windows version of the Paradox Engine's dynamic link library (PXENGWIN.DLL) on your computer or on your network server before you can open Paradox files using QELIB. If you plan to use QELIB to query Paradox files that reside in a shared directory across a network, then you should enter Windows and run the Paradox Engine configuration utility (PXENGCFG.EXE) to set your UserName and your NetNamePath.

The DOS SHARE program must be loaded to access Paradox files.

Connection String

To connect to Paradox database files, the connection string used in qeConnect is:

"DRV=QEPDX"

QLPDX.DLL is the database driver used by QELIB to access Paradox database files.

For Windows, the DLL must either be in your current directory, on your DOS PATH, or in the \WINDOWS\SYSTEM directory.

Multi-User Access to Files

You can use QELIB to access Paradox files on your local computer or on a network server. If the files are on a network server, multiple users can query these files simultaneously.

To share Paradox files among multiple users, the files must be located in a shared directory on your network server. Since the process of establishing shared directories varies from network to network, you should contact your system administrator to set up a shared directory and to configure each user's environment in order to properly access the directory.

Whenever you open a Paradox file that some other user has open at the same time, the consistency of the data becomes an issue if both individuals are updating the file. Before attempting to modify or delete a record, QELIB attempts to put a lock on that record. If the record exists, the operation proceeds accordingly. But if the record has been altered (or already removed) by some other user, QELIB will report an error. All locks are freed as soon as the SQL Insert, Update, or Delete statement has executed.

The QELIB user and the Paradox user can simultaneously edit the same file as long as the Paradox user has selected "Co-Edit" mode.

Create & Drop Table Statements

QELIB supports SQL statements to create and delete Paradox database files. The Create Table statement is used to create files and the Drop Table statement is used to delete files.

Create Table

A sample Create Table statement to create an employee database file is:

```
CREATE TABLE emp.db (last_name CHAR(20),
first_name CHAR(12), salary NUMBER,
hire_date DATE)
```

The form of the Create Table statement is:

```
CREATE TABLE <file_name> (<col_name>
<data_type>, ...)
```

The **file_name** can be a simple file name (emp.db) or a full pathname (C:\QELIB\emp.db). If it is a simple file name, the file is created in the current working directory.

Column names (**col_name**) can contain up to 25 characters. You can use letters, numbers, or the underscore (_) character in names. A name cannot begin with a blank but it can contain a blank. (Names containing blanks must be quoted as shown: 'Name with Blanks'.) The first character of a name must be a letter.

Data_type is the specification of a column's data type. The possible data types in a Create Table statement are:

Data Type	Description
Char	Values can contain letters, numbers, or any of the punctuation keys on your keyboard. A length parameter is required giving the maximum length of a character value that can be stored. The length limit is 254 characters. Example: CHAR(12).
Number	Values can contain only numbers. This includes a decimal point, and optionally, a leading minus sign. There are no parameters. These values are stored as double precision floating point. Example: NUMBER.

Short	Values can contain only whole numbers between -32,767 and 32,767. There are no parameters. Example: SHORT.
Date	Values can contain date values. The time of day is not included. There are no parameters. Example: DATE.
Currency	Values can contain only numbers. Stored the same as Number, but used to represent monetary amounts. There are no paramaters. These values are stored as double precision floating point. Example: CURRENCY.

Drop Table

A sample Drop Table statement to delete the employee database file is:

```
DROP TABLE emp.db
```

The form of the Drop Table statement is:

```
DROP TABLE <file_name>
```

The **file_name** can be a simple file name (emp.db) or a full pathname (C:\QELIB\emp.db). If it is a simple file name, the file must be in the current working directory.

Index Files

An Index is used to read records in sorted order and to improve performance when selecting records and joining files. Paradox indexes are stored in separate files and are either **primary** or **non-primary**.

A primary index is made up of one or more fields from the Paradox file. The primary key fields of a primary index consist of the first N fields of the file (you specify the number of fields, N, when you create the primary index).

Collectively, the primary key fields uniquely identify each record in the Paradox database file. Thus, no two records in a Paradox file can share the exact same values in their primary key fields.

Once a primary index is created for a Paradox database file, the records in that file will be re-ordered based on the primary key fields. At the time a primary index is created, if there are any records whose primary key field values match, Paradox will delete all but the first record from each group of records that duplicate their primary key field values.

If you modify, add, or delete records in the Paradox database file, the primary index will automatically be updated to reflect the changes to the base file. A Paradox database file can have only one primary index. A primary index is a single file with the same name as the Paradox database file on which it is based but with a "PX" extension.

There are two kinds of non-primary indexes: **incsecondary** and **secondary**. The difference between the two is that an incsecondary index is automatically updated whenever the database file is changed, whereas a secondary index is not automatically updated.

Non-primary indexes are defined on any field of the database file. Only one field can be specified for each non-primary index. The values in non-primary indexes are not required to be unique. Thus, two or more records in a Paradox file can share the exact same value in their non-primary key field.

A Paradox database file can have more than one non-primary index so long as each one is based on a different field. Since a non-primary index is dependent on the primary index for a given database file, you must always create a primary index before creating a non-primary index. Each non-primary index consists of two files. One file has a "XNN" extension while the other has a "YNN" extension (where "NN" is the hexadecimal field number of the key field for the index).

Improving Record Selection Performance

Assume you have an employee database file whose first field is emp_id and the primary index is defined on emp_id. When you execute a Select statement that does not contain a Where clause or an Order By clause, QELIB automatically returns employee records ordered by emp_id.

If you have a Where clause that refers to the emp_id column, such as EMP_ID > "E10297", QELIB returns the matching records much faster than if no primary index existed. Since the emp_id field is the primary key field for the primary index, no two records in the employee database file can have the same emp_id value.

Assume you have two incsecondary indexes on the employee file; one on the last_name field the other on the salary field. When you execute a Select statement with no Where or Order By clauses, QELIB returns the employee records ordered by emp_id, since that is the primary key field.

If you add a Where clause on the last_name column, such as last_name = "Woltman", QELIB uses an incsecondary index to find all the employees whose last name is "Woltman" much faster than it would if there were no incsecondary index on last_name.

Likewise, if your Where clause is salary > 30000.00, QELIB uses an incsecondary index on salary to find all the employees whose salary is greater than $30,000.00. If an incsecondary index is used, the records will be returned in the order of the incsecondary index used.

When you execute a Select statement on a Paradox database file, QELIB automatically opens all related index files. QELIB closes all index files when qeEndSQL is called.

Improving Join Performance

When joining database files, index files can greatly improve performance. For a Select statement such as:

```
SELECT * FROM dept.db, emp.db WHERE
dept.dept_id = emp.dept
```

QELIB attempts to use an index on the dept field of the emp.db file. QELIB will not use any indexes on the dept.db file in this example.

To improve join performance, you need an index on the join field of the second file in the From clause. If there is a third file in the from clause, QELIB also uses an index on the field in the third file that joins it to the second file.

Create & Drop Index Statements

QELIB supports SQL statements to create and delete indexes. The Create Index statement is used to create indexes and the Drop Index statement is used to delete indexes.

Create Index Statement

The Create Index statement for Paradox files has the form:

```
CREATE INDEX indexmode ON pathname (
column [, column, ...] )
```

The **indexmode** indicates the type of index to be created. Indexmode must be one of the following keywords: PRIMARY, SECONDARY, or INCSECONDARY. The types of indexes are defined in the previous section.

The **pathname** is the name of the database file. Following **pathname** is a list of **column** names that make up the index. Primary indexes can contain more than one column name, but the first column must be the first field in the file, the second column the second field, etc. Primary indexes must be made up of the first N fields in the database file. Incsecondary and secondary indexes can only contain one column name.

In general, you should create incsecondary indexes rather than secondary indexes. This is because secondary indexes are not updated when records are added, updated, or deleted from the database file. On the other hand, incsecondary indexes are automatically updated when the database file is modified.

Drop Index Statement

There are two forms of the Drop Index statement for Paradox files. The Drop Index statement for a **primary** index has the form:

```
DROP INDEX PRIMARY ON pathname
```

The **pathname** is the name of the database file. When a primary index is dropped, all associated secondary and incsecondary indexes are also dropped automatically.
The Drop Index statement for a **non-primary** index has the form:

```
DROP INDEX indexmode ON pathname ( column
)
```

The **indexmode** is one of the keywords SECONDARY or INCSECONDARY, indicating the type of non-primary index to be dropped. The **pathname** is the name of the database file. The **column** name of the key field for the index.

Select Statement

You use the SQL Select statement to specify the columns and records to be read. The Select statement for Paradox Files is very similar to the Select statement for dBASE files. See the dBASE section in this chapter for additional examples. This section contains the information that is unique to Paradox files.

General Form

The form of the Select statement for Paradox files is:

```
SELECT  * | <col_expr>[, <col_expr>...]
FROM <file_spec>[, <file_spec>...]
[ WHERE <conditions> ]
[ ORDER BY <sort_expr>[, <sort_expr>...]
]
```

Select Clause

Follow SELECT with a list of column expressions you want to retrieve, or just an asterisk (*) to retrieve all columns.

The most common expression is simply a field name (e.g. last_name). More complex expressions can include mathematical operations or string manipulation.

You can list any number of column expressions as long as you separate them by commas. Field names can be prefixed with the file name or table alias (for example, emp.last_name). If more than one file is present in the From clause, you can use "SELECT file.*" to retrieve all fields from a file.

FROM Clause

Follow FROM with a list of file specifications. File specifications have the form:

```
[pathname [table_alias]]
```

The **pathname** can be a simple file name (emp.db) or a complete pathname (c:\qelib\emp.db). If a simple file name is given, the file must be in the current working directory. The .DB extension is not required, QELIB automatically adds the extension if it is not present.

The **table_alias** is a name used to refer to this file in the rest of the Select statement. Database field names can be prefixed by the table alias. Given the file specification:

```
FROM emp.db E
```

you can refer to the last_name field as E.last_name. You must use table aliases if the Select statement joins a table to itself. For example:

```
SELECT * FROM employee.db E, employee.db F
WHERE E.mgr_id = F.emp_id
```

Where Clause

The Where clause specifies the conditions that records must meet to be retrieved. The Where clause contains conditions in the form:

```
<expr1> <rel_operator> <expr2>
```

exp1 and **exp2** can be field names, constant values, or expressions. **rel_operator** is the relational operator that links the two expressions.

For example, the Select statement used to retrieve the names of employees that make at least $20,000 is:

```
SELECT last_name,first_name FROM emp.db
WHERE salary >= 20000
```

Order By Clause

The Order By clause indicates how the records are to be sorted. The form of the clause is:

```
ORDER BY <sort_expression> [DESC | ASC],
...
```

The **sort_expression** can be field names, expressions, or the number of the column expression to use.

As an example, to sort by last_name, you could use either of the following Select statements:

```
SELECT emp_id, last_name, first_name FROM
emp.db ORDER BY last_name

SELECT emp_id, last_name, first_name FROM
emp.db ORDER BY 2
```

In the second example, last_name is the second column expression following Select, so Order By 2 sorts by last_name.

SQL Expressions

Expressions are used in the column expressions, Where clauses, and Order By clauses of SQL Select statements.

Expressions allow you to use mathematical operations as well as character string and date manipulations to form complex database queries.

QELIB's Paradox support provides a rich set of operators and functions you can use in expressions. The values in expressions can come from fields of the database records, or constant values.

Constants

Constants are values which do not change. For example, in the expression PRICE * 1.05, the value 1.05 is a constant.

You must enclose character constants in pairs of single (') or double quotes ("). To include a single quote in a character constant enclosed by single quotes, use two single quotes together, for example, 'Don''t'. Similarly, if the constant is enclosed by double quotes, use two double quotes to include one.

You must enclose date constants in braces ({}), for example, {01/30/89}. Date constants must be in the form MM/DD/YY or MM/DD/YYYY.

Numeric Expressions

You can include these operators in numeric expressions:

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
^	Exponentiation

The following shows examples of numeric expressions. For these examples, assume SALARY is 20000:

Example	Resulting value
SALARY + 10000	30000
SALARY * 1.1	22000
2 ** 3	8

You can precede numeric expressions with a unary plus (+) or minus (-). For example -(SALARY * 1.1) is -22000.

Character Expressions

Character expressions can include the following operators:

Operator	Meaning
+	Concatenation keeping trailing blanks.
-	Concatenation moving trailing blanks to the end.

The following shows examples of character expressions. In the examples, LAST_NAME is 'BENNETT ' and FIRST_NAME is 'TYLER ':

Example	Resulting value
FIRST_NAME + LAST_NAME	'TYLER BENNETT '
FIRST_NAME - LAST_NAME	'TYLERBENNETT '
FIRST_NAME - (' ' + LAST_NAME)	'TYLER BENNETT '

Date Expressions

You can include the following operators in date expressions:

Operator	Meaning
+	Add a number of days to a date to produce a new date.
-	Show the number of days between two dates, or subtract a number of days from a date to produce a new date.

The following shows examples of date expressions. In these examples, HIRE_DATE is {01/30/90}.

Example	Resulting value
HIRE_DATE + 5	{02/04/90}
HIRE_DATE - {01/01/90}	29
HIRE_DATE - 10	{01/20/90}

Relational Operators

The relational operators (**rel_operator**) separating two expressions can be:

Operator	Meaning
=	Equal
<>	Not Equal
!=	Not Equal
#	Not Equal
>	Greater Than
>=	Greater Than or Equal
!<	Not Less Than (same as >=)
<	Less Than
<=	Less Than or Equal
!>	Not Greater Than (same as <=)
LIKE	Matching a pattern
NOT LIKE	Not matching a pattern
*=	Outer Join, Matches NULL for second value
=*	Outer Join, Matches NULL for first value
IS NULL	Equal to NULL
IS NOT NULL	Not Equal to NULL
BETWEEN	Values between lower and upper bound

Examples

```
SALARY <= 40000
DEPT = 'D101'
HIRE_DATE > {01/30/89}
SALARY + COMMISSION  >= 50000
LAST_NAME LIKE 'Be%'
SALARY IS NULL
SALARY BETWEEN 10000 AND 20000
```

Logical Operators

You can combine two or more conditions, related by an AND or OR, to form complex criteria. For example:

```
SALARY = 40000 AND EXEMPT = 1
```

Use the logical NOT operator to reverse the meaning:

```
NOT (SALARY = 40000 AND EXEMPT = 1)
```

Operator Precedence

As expressions become more complex, the order in which the expressions are evaluated becomes important. The following precedence file shows the order in which the operators are evaluated. The operators in the first line are evaluated first, then those in the second line, etc. Operators in the same line are evaluated left to right in the expression.

Precedence	Operators
1	Unary -, Unary +
2	**, ^
3	*, /
4	+, - (between numbers, dates, or character strings)
5	=, <>, !=, #, <, <=, !<, >, >=, !>, *=, =*, LIKE, NOT LIKE, IS NULL, IS NOT NULL, BETWEEN,
6	NOT
7	AND
8	OR

The following example shows the importance of precedence:

```
WHERE SALARY > 40000 OR
    HIRE_DATE > {01/01/89} AND
    DEPT = 'D101'
```

Because AND is evaluated first, this query retrieves employees in department D101 hired after Jan 1, 1989, as well as every employee making more than $40,000, no matter what department or hire date.

To force the clause to be evaluated in a different order, use parentheses to enclose the conditions to be evaluated first. For example:

```
WHERE (SALARY > 40000 OR HIRE_DATE >
{01/01/89}) AND DEPT = 'D101'
```

retrieves employees in department D101 that either make more than $40,000 or were hired after Jan 1, 1989.

Functions

QELIB supports several Paradox functions to use in expressions. In the following sections, functions are grouped according to the type of result they return.

Functions that Return character Strings

Function	Meaning
CHR	Converts an ASCII code into a one-character string. CHR(67) returns 'C'.
UPPER	Uppercase each letter of a string. UPPER('Rappl') returns 'RAPPL'.
LOWER	Lowercase each letter of a string. LOWER('Rappl') returns 'rappl'.
SUBSTR	Substring of a string. Parameters are the string, the first character to extract, and the number of characters to extract. SUBSTR('Holcomb',2,3) returns 'olc'.
SPACES	Generate a string of blanks. SPACES(5) returns ' '.
STRVAL	Convert a value of any type to a character string. STRVAL('Woltman') returns 'Woltman'. STRVAL ({12/25/53}) returns '12/25/53'. STRVAL (5 * 3) returns '15'. STRVAL (4 = 5) returns 'False'.

TIME	Returns the time of day as a string. At 9:49 PM, TIME() returns '21:49:00'.
USERNAME	Returns the name of the current user as a string. USERNAME() might return 'Bennett'.

Functions that Return Numbers

Function	Meaning
MOD	Divides two numbers, returning the remainder of the division. MOD(10,3) returns 1.
LEN	The length of a string. LEN('ABC') returns 3.
MAX	Returns the larger of two numbers. MAX(66,89) returns 89.
MIN	Returns the smaller of two numbers. MIN(66,89) returns 66.
POW	Raises number to a power. POW(3,2) returns 9
INT	Returns the integer of a number. INT(6.4321) returns 6.
ROUND	Rounds a number to a specified number of digits. ROUND(123.456, 0) returns 123. ROUND(123.456, 2) returns 123.46. ROUND(123.456, -2) returns 100.
MONTH	The month part of a date. MONTH({01/30/89}) returns 1.
DAY	The day part of a date. DAY({01/30/89}) returns 30.
NUMVAL	Converts a character string to a number. The parameter is a character string. If the parameter is not a valid number, a zero is returned. NUMVAL("123") returns the number 123.
YEAR	The year part of a date. YEAR({01/30/89}) returns 1989.

Functions that Return Dates

Function	Meaning
TODAY	Today's date. If today is 12/25/79, TODAY() returns {12/25/79}.
DATEVAL	Convert a character string to a date. DATEVAL('01/30/89') returns {01/30/89}.

The following examples use the Paradox functions:

Retrieve all employees that have been with the company at least 90 days:

```
SELECT FIRST_NAME,LAST_NAME FROM EMP WHERE
TODAY() - HIRE_DATE  >= 90
```

Retrieve all employees hired in January of this year or in January of last year:

```
SELECT FIRST_NAME,LAST_NAME FROM EMP WHERE
MONTH(HIRE_DATE) = 1 AND (YEAR(HIRE_DATE)
= YEAR(TODAY()) OR YEAR(HIRE_DATE) =
YEAR(TODAY()) - 1)
```

Insert Statement

The SQL Insert statement adds new records to a database file. An example of an Insert statement on the employee file is:

```
INSERT INTO emp.db (last_name, first_name,
emp_id, salary, hire_date) VALUES
('Smith', 'John', 'E22345', 27500,
{4/6/91})
```

Each Insert statement adds one record to the database file. In this case, a record has been added to the employee database file, emp.db. Values are specified for five columns and the remaining columns are assigned a blank (Null) value.

Note

Character values must be enclosed in quotation marks and dates must be enclosed in braces {}.

The form of the Insert statement supported for Paradox files is:

```
INSERT INTO <file_name> [(<col_name>,...)]
VALUES (<expr>, ...)
```

The column names are optional.

The **file_name** is the name of the database file QELIB adds the record to.

The **col_name** list is an optional list of column names giving the name and order of the columns whose values are specified in the Values clause. If you omit col_name, the value expressions (**expr**) must be in the order the columns are defined for the file.

The **expr** list is the list of expressions giving the values for the columns of the new record. Usually, the expressions are constant values for the columns. If you specify the col_name list, the values must be in the order the columns are listed. If the col_name list is omitted, the values must be in the order of the columns in the file. Character string values must be enclosed with single or double quote characters, and date values must be enclosed by braces {}.

Update Statement

The SQL Update statement is used to change records in a database file. An example of an Update statement on the employee file is:

```
UPDATE emp.db SET salary=32000, exempt=1
WHERE emp_id = 'E10001'
```

The Update statement changes every record that meets the conditions in the Where clause. In this case the salary and exempt status has been changed for all employees having the employee ID E10001. Since employee ID's are unique in the employee file, one record is updated.

The form of the Update statement supported for Paradox files is:

```
UPDATE <file_name> SET <col_name> =
<expr>, ... [ WHERE <conditions> ]
```

The Where clause is optional.

The **file_name** is the name of the database file to be updated.

col_name is the name of a column whose value is to be changed. Several columns can be changed in one statement.

The **expr** is the new value for the column. Usually, the expression is a constant value. Character string values must be enclosed with single or double quote characters, and date values must be enclosed by braces {}.

The Where clause is any valid clause for Paradox files. It determines which records are to be updated.

Delete Statement

The SQL Delete statement is used to delete records from a database file. An example of a Delete statement on the employee file is:

```
DELETE FROM emp.db WHERE emp_id = 'E10001'
```

Each Delete statement removes every record that meets the conditions in the Where clause. In this case the every record having the employee id E10001 is deleted. Since employee ID's are unique in the employee file, one record is deleted.

The form of the Delete statement supported for Paradox files is:

```
DELETE FROM <file_name> [ WHERE
<conditions> ]
```

The Where clause is optional.

The **file_name** is the name of the database file whose records are to be deleted.

The Where clause is any valid clause for Paradox files. It determines which records are to be Deleted.

Commit and Rollback

The Paradox driver immediately executes Insert, Update, and Delete statements on the database files and the changes are automatically committed when the SQL statement is executed, freeing all locks.

qeBeginTran, qeCommit, and qeRollback cannot be used with Paradox files.

Data Types

The following table shows how the Paradox data types are mapped to the eight standard QELIB data types returned by qeColType, and to the underlying database types returned by qeColDBType.

Paradox	qeColType	qeColDBType
Char	Variable length character string, type 2	2
Number	Double float, type 7	7
Short	Integer, type 5	5
Currency	Double float, type 7	116
Date	Date-time, type 8	111
Numeric Expr	Double float, type 7	7
Char Expr	Variable length character string, type 2	2

Numeric Expr and Char Expr are used for expressions. For example, in:

```
SELECT salary*1.1, last_name+first_name
FROM emp.db
```

salary*1.1 is a numeric expression, and last_name+first_name is a character expression.

Index

E

F

H

I

L